



Calhoun: The NPS Institutional Archive
DSpace Repository

Theses and Dissertations

1. Thesis and Dissertation Collection, all items

1987-06

Comparison of Pascal and the dBASE III PLUS
language in programming an inventory
management system

Chang, To.

<http://hdl.handle.net/10945/22676>

Downloaded from NPS Archive: Calhoun



Calhoun is the Naval Postgraduate School's public access digital repository for research materials and institutional publications created by the NPS community. Calhoun is named for Professor of Mathematics Guy K. Calhoun, NPS's first appointed -- and published -- scholarly author.

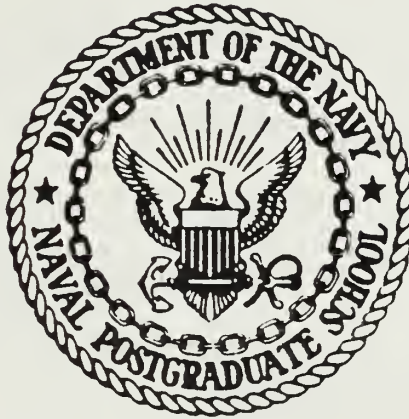
Dudley Knox Library / Naval Postgraduate School
411 Dyer Road / 1 University Circle
Monterey, California USA 93943

<http://www.nps.edu/library>

DEPT. OF NAVY
NAVAL OFFICE
MONTEREY, CALIF. 94038-3002

NAVAL POSTGRADUATE SCHOOL

Monterey, California



THESIS

COMPARISON OF PASCAL AND THE
DBASE III PLUS LANGUAGE IN PROGRAMMING
AN INVENTORY MANAGEMENT SYSTEM

by

To Chang

June 1987

Thesis Advisor:

C. Thomas Wu

Approved for public release; distribution is unlimited.

T233163

unclassified

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1 REPORT SECURITY CLASSIFICATION unclassified			1b RESTRICTIVE MARKINGS		
2 SECURITY CLASSIFICATION AUTHORITY			3 DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution is unlimited.		
5 DECLASSIFICATION/DOWNGRADING SCHEDULE					
PERFORMING ORGANIZATION REPORT NUMBER(S)			5 MONITORING ORGANIZATION REPORT NUMBER(S)		
1 NAME OF PERFORMING ORGANIZATION Naval Postgraduate School		6b OFFICE SYMBOL (if applicable) 52	7a NAME OF MONITORING ORGANIZATION Naval Postgraduate School		
ADDRESS (City, State, and ZIP Code) Monterey, California 93943-5000			7b ADDRESS (City, State, and ZIP Code) Monterey, California 93943-5000		
NAME OF FUNDING/SPONSORING ORGANIZATION		8b OFFICE SYMBOL (if applicable)	9 PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER		
ADDRESS (City, State, and ZIP Code)			10 SOURCE OF FUNDING NUMBERS		
			PROGRAM ELEMENT NO	PROJECT NO	TASK NO
			WORK UNIT ACCESSION NO		
TITLE (Include Security Classification) COMPARISON OF PASCAL AND THE DBASE III PLUS LANGUAGE IN PROGRAMMING AN INVENTORY MANAGEMENT SYSTEM					
PERSONAL AUTHOR(S) Chang, To					
TYPE OF REPORT Master's Thesis		13b TIME COVERED FROM TO	14 DATE OF REPORT (Year, Month, Day) 1987 June		15 PAGE COUNT 203
SUPPLEMENTARY NOTATION					
COSATI CODES			18 SUBJECT TERMS (Continue on reverse if necessary and identify by block number)		
FIELD	GROUP	SUB-GROUP	database management systems (DBMS); inventory management program; dBase III PLUS		
ABSTRACT (Continue on reverse if necessary and identify by block number) Before the widespread use of Database Management Systems (DBMS), pro- grammers have had to rely on the third generation language such as COBOL, Pascal, and PL/I to implement their application programs. These programs are usually very hard to maintain and modify unless very disciplined structured programming techniques are used. However, with the DBMS, the ease of development, maintenance, and modification of data-managing application programs can be attained. In this thesis, we compare two versions of an inventory management program, one written in Pascal and the other written in dBASE III PLUS, in terms of their modifiability and maintainability.					
DISTRIBUTION/AVAILABILITY OF ABSTRACT ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS			21 ABSTRACT SECURITY CLASSIFICATION unclassified		
NAME OF RESPONSIBLE INDIVIDUAL Prof. C. Thomas Wu			22b TELEPHONE (Include Area Code) (408) 646-3391	22c OFFICE SYMBOL Code 52Wq	

Approved for public release; distribution is unlimited.

Comparison of Pascal and the dBASE III PLUS language
in Programming an Inventory Management System

by

Chang, To
Major, Republic of China Marine Corps
B.S., Chinese Naval Academy, 1979

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
June 1987

ABSTRACT

Before the widespread use of Database Management Systems (DBMS), programmers have had to rely on the third generation language such as COBOL, Pascal, and PL/I to implement their application programs. These programs are usually very hard to maintain and modify unless very disciplined structured programming techniques are used. However, with the DBMS, the ease of development, maintenance, and modification of data-managing application programs can be attained. In this thesis, we compare two versions of an inventory management program, one written in Apple Pascal™ and the other written in dBASE III PLUS™, in terms of their modifiability and maintainability.

18/05
37/8
C.I

TABLE OF CONTENTS

I.	INTRODUCTION.....	8
II.	THE PROGRAMMING PRINCIPLE OF PASCAL.....	12
	A. INTRODUCTION.....	12
	B. NAME AND CONTROL STRUCTURES.....	13
	C. DATA STRUCTURES.....	14
	1. Real and Integer.....	14
	2. Boolean Variables, Expressions and Operators....	15
	3. Characters Variables.....	16
	4. Programmer-Defined Data Types.....	16
	5. Set Types.....	19
	6. Array Types.....	20
	7. Record Types.....	20
	8. Pointer Types.....	22
	D. SUMMARY.....	24
III.	THE PROGRAMMING PRINCIPLE OF dBASE III PLUS.....	26
	A. AN OVERVIEW OF dBASE III PLUS.....	26
	B. THE dBASE III PLUS SYSTEM.....	27
	C. DATA TYPES OF dBASE III PLUS.....	27
	1. Character Data Type.....	28
	2. Numeric Data Type.....	29
	3. Logical Data Type.....	29
	4. Date Data Type.....	30
	5. Memo Data Type.....	30
	D. DATA STRUCTURES OF dBASE III PLUS.....	30
	1. The Data File Structure.....	31

2.	Indexing.....	32
E.	PROGRAMMING IN dBASE III PLUS.....	33
1.	Parameter Passing.....	33
2.	Control Transfers From One program To Another	35
F.	HOW DOES dBASE III PLUS IMPLEMENT THE RELATIONAL MODEL.....	35
G.	I/O PROCESSING OF dBASE III PLUS.....	36
1.	Output.....	37
2.	Input.....	39
H.	SUMMARY.....	41
IV.	STUDY AND ANALYSIS OF ORIGINAL PROGRAM.....	43
A.	BACKGROUND.....	43
B.	ENTITY-RELATIONAL DIAGRAM.....	43
C.	PROGRAM STRUCTURES.....	45
1.	Sale.....	45
2.	Purchase.....	52
V.	ANALYSIS AND DESIGN OF NEW PROGRAM.....	58
A.	NORMALIZATION OF SALES.....	59
1.	1st Normal Form.....	60
2.	2nd Normal Form.....	61
3.	3rd Normal Form.....	62
4.	New E-R Diagram.....	63
5.	New Data Structures.....	63
B.	NORMALIZATION OF PURCHASE.....	66
1.	1st Normal Form.....	67
2.	2nd Normal Form.....	68
3.	3rd Normal Form.....	68

4.	New E-R Diagram.....	69
5.	New Data Structures.....	69
VI.	STUDY OF MAINTANIABILITY OF THESE TWO PROGRAMS..	72
A.	THE ORIGINAL PROGRAM.....	73
1.	Maintainability of Record Structures.....	73
B.	THE NEW PROGRAM.....	78
1.	Maintainability of Record Structures.....	79
VII.	STUDY OF MODIFIABILITY OF THESE TWO PROGRAMS.....	83
A.	MODIFIABILITY OF PASCAL.....	83
1.	Modifying Data Fields in Pascal Program.....	83
2.	Modifying Functions in Pascal Program.....	85
B.	MODIFIABILITY OF dBASE III PLUS.....	86
1.	Modifying Data Fields in dBase III PLUS Program..	86
2.	Modifying Functions in dBase III PLUS program..	88
VIII.	CONCLUSION.....	91
	APPENDIX A: THE ORIGINAL PROGRAM.....	93
	APPENDIX B: THE NEW PROGRAM.....	158
	LIST OF REFERENCES.....	199
	INITIAL DISTRIBUTION LIST.....	201

LIST OF FIGURES

2.1	Integer and Real Format.....	15
2.2	A Sample of Multiple-linked List.....	24
2.3	A Sample Tree.....	24
3.1	A Conceptual View of dBase III PLUS Data File.....	32
4.1	E-R Diagram of Original Program.....	44
4.2	Structured Diagram of SALES.....	45
4.3	Structured Diagram of FSSNEW.....	47
4.4	Structured Diagram of FSSSHIPMENT.....	48
4.5	Structured Diagram of FSSINQUERY.....	50
4.6	Structured Diagram of PURCHASE.....	52
4.7	Structured Diagram of FSPNEW.....	53
4.8	Structured Diagram of FSPSHIPMENT.....	55
4.9	Structured Diagram of FSPINQUERY.....	57
5.1	New E-R Diagram of SALES.....	65
5.2	New E-R Diagram of PURCHASE.....	70

I. INTRODUCTION

In the early 1960's, when database processing was considered an esoteric subject, data was organized in a sequential manner. Where physical structure and logical structure are identical, and data were sent into computer as batch processing without real-time access. In this case, multiple copies of the same files are kept. At that time, the software handled the I/O operations. If the physical structure changed, application programs need to be rewritten, recompiled and retested. Data was designed and optimized for a single application, there was a high level of program/data dependence.

Late in the 1960's, both serial and random access to records was possible. The logical and physical layout of such files are distinct, but the relationship between them is simple. Now, data storage units can be changed without changing the application program. Data structure is usually designed as sequential, indexed sequential, or simple direct access. Multiple key retrieval is generally not used. Data security measures can be used but are likely to be very elementary. Still, much data redundancy exists. In this stage, software provides data access methods but not data management.

In the early 1970's, multiple logical files can be derived from the same data, and those data can be accessed in different ways by applications with different requirements. Data elements are shared between diverse applications. The absence of redundancy facilitates data integrity. Application storage organization is

independent of the application program. It can be changed to improve database performance without affecting application programs. Multiple key retrieval can be used where complex of programs. The program in this study written in the Apple Pascal data organization are used without complicating application language (see Appendix A) falls into this category.

Now, at the current stage of database processing, software provides logical as well as physical data independence. Data can evolve without incurring high maintenance costs. Utilities are provided so that a database administrator can act as controller and custodian of the data to insure that its organization is least for the users as a whole. Effective procedures are provided for controlling privacy, security and integrity of the data. With these, the database can easily provide answers to unanticipated requests. More than that, a data description language is provided for the database administrator. Also, a command language exists for the application programs, and a query language exists for the casual user. The dBASE III PLUS language (see Appendix B) falls into this category.

An amazing amount of progress has been made in the computer field since the primitive computer age of the 1950s. Personal computers, high-level languages, artificial intelligence, and many other technological advancements have been made in a period of only 35 years, and new applications are being discovered every day.

The idea of recording and maintaining information in an organized manner appeared many years ago, when the value of organized information was realized. The importance of this idea is stressed in the Spinoza expression : "The order and connection of ideas is the same as the order and connection of things". However,

the appearance of computers started enforcing this idea with the implementation of applications on the computer.

The use of automation and parallelism theories has also helped the designers to make retrieval of very large databases very easy, and in extremely timely manner.

The tremendous progress in the database design has resulted in lower cost, and has provided a strong motivation for working in the database development field, especially on every large database.

An additionally strong motivation for working in the database field is the wide variety of database applications. These applications include manufacturing with inventory management, the servicing of industries with lists of service capabilities; economic models with production data for allocation and planning, and medical services with patient records, disease histories, problem classification, and treatment effectiveness data. Thus, database are appearing and supporting almost every science. It might be said that it is the database era in computer application.

An important consideration in the design of the database is the way of storing data, which is used for a broad variety of application and can be used to make changes to the data quickly and easily. The ability of the database to be applicable in so broad an aspect of applications is based on a common feature that makes database development valuable and general in a programming methodology. This feature is a creative form which is called "structural growth". This "structural growth" should start with a solution on a simplified version of the problem and then repeatedly expand its capabilities up to desired level.

Database systems are now available on machines that range all the way from quite small microcomputers¹⁴ to the largest mainframes. The facilities provided by any given system are to

some extent determined by the size and power of the underlying machine.

Following will be the detailed discussion of both the old program written in Apple Pascal which ran on Apple II PLUS, and the new program written in dBASE III PLUS which is going to run on IBM PC, followed by a study of the maintainability and modifiability of these two programs.

II. THE PROGRAMMING PRINCIPLE OF PASCAL

A. INTRODUCTION

The development of Pascal began in 1968 and resulted in a compiler written entirely in Pascal in 1970 by Professor Nicklaus Wirth of Zurich, Switzerland. The language was slightly revised in 1972 and is undergoing standardization efforts. It has become very popular as a language for teaching programming and is widely used on microcomputers. Its popularity is due to the fact that its syntax is relatively easy to learn. Also, Pascal facilitates writing **structured programs** - programs that are relatively easy to read, understand, and maintain. It is an Algol-like language, but unlike Algol's key words, Pascal's reserved words are not typed differently from identifiers.

In Algol, there are three primitive data types, and Booleans. These, in turn, were very similar to the primitive data types provided by FORTRAN. This reflects the fact that both of these languages are predominantly scientific programming languages. Numbers and logical values are the most useful objects for scientific programming. Pascal extends its applicability to commercial and systems programming by providing one additional primitive data type, CHARACTERS. Pascal is a third generation language, and a reaction to the second generation languages. Its emphasis is on simplicity and efficiency. There are two similar standards for Pascal. They are **ANSI/IEEE** (American National Standards Institute/Institute for Electrical and Electronics Engineers) Standard, and **ISO** Inter- national Standard.

Although Pascal was intended as a teaching language, many other programmers have found that it is also suitable for "real"

programming. Its strong typing simplifies debugging and helps catch latent errors in production programs; its rich set of efficient, high level data types simplifies many non-numeric programs; and its small size means that a programmer can acquire mastery of the language in a moderate amount of time.

These qualities have made Pascal an attractive vehicle for programming research. Pascal has been extended for concurrent programming, to support verification, and for operating system writing, or even a database writing (the original program in this thesis is a good example of it). Pascal has become a basis for almost all new language designs; most new languages are "Pascal-like." This includes the language Ada.

B. NAME AND CONTROL STRUCTURES

Pascal includes important additions to Algol's name, data, and control structuring mechanisms. Variable declarations are introduced by the word **var** and have the syntax:

<names> : <type>

Procedure and function declarations are quite similar to Algol's, except that the **begin** comes after the local declarations rather than before them:

procedure <name> (<formals>);

< declarations >

begin

< statements >

end;

In addition to variable and procedure declarations, Pascal has constant and type bindings. Variables can be declared to be the type of a range of integer. This new data type then can be used in other data types. Type declarations are introduced by the word

type and have the syntax:

< name > = < type >

Pascal has added a character data type for nonnumeric programming and a variety of data type constructors for arrays, records, sets, pointers, and so forth. Programmers can use these, in conjunction with type declarations, to design data types specifically suited to their applications. In the program HANAOKA, a lot of these techniques are used.

Pascal's control structures incorporate many of the ideas of structured programming. Of course the **if-then-else** and **for-loop** (in a very simplified form) are provided. Pascal also provides leading and trailing decision loops and a **case**-statement for handling the breakdown of a problem into many cases. The **goto** is provided in a simplified form.

C. DATA STRUCTURES

Pascal inherits the three primitive data types from Algol: reals, integers, and Booleans. These are considered to be the standard data types (or simple data types) of Pascal.

1. Real and Integer

The data types INTEGER and REAL are used to represent numeric information. People use INTEGER variables as loop counters and to represent data such as an exam score or those without decimal point. The data type REAL can be used to represent all numbers, as a matter of fact, INTEGER is a subset of REAL. On many computers though operations involving integers are faster and less storage space is needed to store integers. Also operations with integers are always precise whereas there may be some loss of accuracy when dealing with real numbers.

These differences result from the way real numbers and integers are represented internally in memory. real-numbers tend to be computer dependent; some sample integer and real formats are shown in Fig. 2-1.

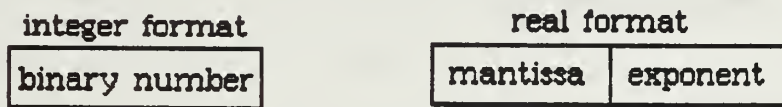


Figure 2-1 Integer and Real Format

In Fig. 2-1, each integer is represented as a standard binary number. Real format is analogous to scientific notation. The storage area occupied by a real number is divided into two sections: the **mantissa** and the **exponent**. All the arithmetic operators (+, -, *, /) seen so far can be used with either integer or real operands. But the two operators, **div** and **mod**, that must be used only with type INTEGER operands. With these operators, we can write multiple-operator expressions that compute the desired results.

2. Boolean Variables, Expressions and Operators

A BOOLEAN variable or constant can be set to either of the BOOLEAN values, TRUE or FALSE. The statement

```
const  
    GOOD = TRUE;
```

specifies that the BOOLEAN constant GOOD has the value TRUE. We can use the relational operators (=, <, >, etc.) with numeric data to form conditions or BOOLEAN expressions. There are three BOOLEAN operators: **and**, **or**, **not**. These operators are used with operands that are BOOLEAN expressions. BOOLEAN variables can be used as program flags to signal whether or not a special event

occurred in a program. The fact that such an event occurred is important to the future execution of the program. A BOOLEAN variable used as a program flag is initialized to one of its two possible values (TRUE or FALSE) and reset to the other as soon as the event being monitored occurs.

3. Characters Variables

Pascal provides a character data type that can be used for the storage and manipulation of the individual characters that comprise a person's name, address, etc. Character variables are declared using the data type **CHAR** in a declaration. A character value consists of a single printable character (letter, digit, punctuation mark, etc.) enclosed in apostrophes. A character value can be assigned to a character variable or associated with a constant identifier.

Relational operators can be used with characters. For example, the BOOLEAN expressions

SENTENCE = BLANK

SENTENCE <> PERIOD

are used to determine whether two character variables have the same value or different values. Order comparisons can also be performed on character variables using the relational operators <, <=, >, >=.

4. Programmer-Defined Data Types

One of the features of Pascal that accounts for its widespread use is that it permits the declaration of new data types. In Pascal, you can define **enumerated types**, **subrange types** and **set types**.

Often programs must manipulate nonnumeric data; this is

usually character data, but it can also be more abstract. For example, a commercial data-processing program may need to be able to deal with days of the week. With **enumerated types** we can construct types by enumerating, or listing, all their possible values. For example, we can declare the types for months, days of the week, and sexes like this:

type

```
month = (Jan, Feb, Mar, Apr, May, Jun, Jul, Aug, Sep, Oct,
         Nov, Dec);
DayOfWeek = (Sun, Mon, Tue, Wen, Thu, Fri, Sat);
sex = (male, female);
```

It is then possible to declare variables of these types and use them:

var

```
today, tommorow : DayOfWeek;
ThisMonth : month;
gender : sex;
```

begin

```
.....
today := Tue;
today := tommorow;
ThisMonth := Apr;
gender := female;
```

Pascal also preserves security by preventing the programmer from performing meaningless operations on enumeration values. People use **abstract data type**: a set of data values and the primitive operations on those data values. For an enumerated type, the set of data values are specified in the enumeration. The operations don't have to be specified because they are the same for all enumerated types:

```
:=, succ, pred
=, <>, <, >, <=, >=
```

The ordering relations (<,>, etc.) are defined according to the order specified in the declaration of the enumerated type. For example, Mon < Wen and Dec > Jan. The **succ** and **pred** functions give the succeeding and preceding elements in the list. For example, succ(Mon) = Tue and pred(Mar) = Feb. These operations are also secure; for example, succ(Sat) and pred(Jan) are errors.

The benefits of enumerated types can be summarized as followed:

1. They are high level and application oriented.
2. They allow programmers to say what they mean.
3. They are efficient since they allow the compiler to economize on storage, and the operations can be performed quickly.
4. They are efficient since the compiler ensures that programmer can't do meaningless operations.

We have seen that the enumerated type improves security since the compiler can check if the programmer is doing something meaningless, such as asking for the successor of the last element in the enumeration. The Pascal **subrange type constructor** extends this checking to integers and allows tighter checking on other types. Suppose the variable DayOfMonth is used to hold meaningful values from 1-31. although this could be declared as an integer variable, our program will be more secure if we use a subrange type:

```
var DayOfMonth : 1..31;
```

If we attempt to assign to this variable a value outside this range, we will get an error.

Subrange declarations also allow the the compiler to economize on storage utilization. Subrange types can be based on types other than integers.

```
type WeekDay = Mon..Fri;
```

If we accidentally assigned Sat or Sun to a variable of type WeekDay, we would get an error. Also, Pascal permits the programmer to define subranges of any discrete type, that is, enumerated types, integers, and characters. It does not permit defining a subrange of the real numbers, which is a continuous type.

5. Set Types

Pascal provides the ability to manipulate small finite **sets** using the standard operations of set theory. The set type is almost an ideal data type. It is high level and application oriented yet very efficient.

The description of a set type has the form

set of < simple type >

where a <simple type> is an enumerated type (including **char**), a subrange type, or a name of one of these. An existing set can be modified using the set operators. Before a set can be manipulated, its initial elements must be defined using a set assignment statement. A set variable must always be initialized before it can be used with any of the set operators. The set operators union, intersection, and difference require two sets of the same type as operands. The +, *, and - are treated as set operators when their operands are sets. These operators can be used to combine two sets to form a third set. If more than one set operator is used in an expression, the normal precedence rules for the operators +, *, and - will be followed. When in doubt, it is best to use parentheses to specify the intended order of evaluation.

Sets may also be compared through the use of the relational operators =, <=, etc. Both operands of a set relational operator must have the same base type.

6. Array Types

Pascal is descendant of Algol-60, and Algol-60 generalizes FORTRAN arrays in two respects: It allows any number of dimensions and it allows lower bounds other than one. Pascal has generalized Algol's arrays in some respects and has restricted them in others.

One of the generalizations is in the allowable **index types**. They can be subscripted by many other types including characters, enumerated types, and subranges of these.

```
var A : array [1..100] of real;
```

Notice that the dimensions of the array have been specified as a subrange of the integers. Actually, any finite discrete type can be used as an index type.

Another way in which Pascal generalizes Algol arrays is in the allowable element types. Now any type can be the base type of an array type. That is, we can have arrays of integers, reals, characters, enumerated types, subranges, records, pointers, and so forth. In general, a Pascal array-type constructor has the form

```
array [ <index type> ] of <base type>
```

Where <index type> is any finite discrete type and <base type> is any type at all. Thus, Pascal arrays can be considered **finite mapping** from the index type to the base type.

Arrays can be defined as multidimensional arrays. Suppose we need a 20X100 arrays of reals M, we can define

```
var M : array [1..20] of array [1..100] of real;
```

As mentioned above, the base type of an array can be any type, including another array type.

7. Record Types

One of the most important data structure constructors provided by Pascal is the **record-type constructor**. This is a data structure that allows arbitrary groups of data.

```
type person = record
    name : string;
    age   : 18..100;
    rank  : string;
    sex   : (male,female)
    birthdate : date;
end;
```

Just like an array, a record has a number of components. Unlike an array, however, the components of a record can be of different types. Also the components of records can themselves be complex data types. The components of arrays are selected by subscripting. A component of a record is selected by placing a period between the name of the record and the name of the component. Selectors for records and arrays can be combined as needed to access a particular component. But why have both arrays and records since they are both methods of grouping data together. They differ in two important respects. Arrays are **homogeneous**, that is, all of the components of an array are the same type. Records are **heterogeneous**, that is, their components do not have to be the same type. In this sense records are more general than arrays.

Since arithmetic and logical operations must be performed on individual memory cells, record variables cannot be used as the operands of arithmetic and relational operators. These operators must be used with individual fields of a record.

The other difference between arrays and records is in their manner of selecting components. We can select specific array

elements with expressions like $A[1]$, $A[2]$ just as we can select specific record components with expressions like $R.mon$, $R.day$. The difference is that we can compute the selector to be used with arrays; that is we can write $A[E]$ where E is an expression whose value will be known at run-time. This is an important feature since it allows writing a loop that process all the elements of an array. This can't be done with records.

8. Pointer Types

Pointer types are dynamic data structure of Pascal. Dynamic data structures are data structures that "grow" as a program executes. A dynamic data structure is a collection of elements (called **nodes**) that are normally records. Unlike an array that always contains storage space for a fixed number of elements, a dynamic data structure expands and contracts during program execution based on the data storage requirements of the program.

Dynamic data structures are used for storage of real world data that is constantly changing. An example would be an airline passenger list.

Dynamic data structures are extremely flexible. It is relatively easy to add new information by creating a new node and inserting it between two existing nodes. It is also relatively easy to modify dynamic data structures by removing or deleting an existing node. This is more convenient than modifying an array of records, where each record is in a fixed position relative to the others as determined by its subscript. Here is an example using this feature:

```
var p : pointer;  
    x : integer;  
begin  
    new(p);
```

```

.....
p^ := 5;
.....
x := x + p;
.....
end;

```

This program allocates a memory location and puts its address in P, stores 5 in the memory location whose address is in P, and then add the contents of this location to x.

Since we don't know beforehand the order or number of **nodes** in a dynamic data structure, we cannot allocate storage from a dynamic data structure in the conventional way (using a variable declaration statement). Instead, we must allocate storage for each individual node as needed and join this node to the rest of the structure. The **new** statement is used to allocate storage for a new node.

Also we must have some way of referencing each new node that is allocated in order to store data in it. Pascal provides a special type of variable, called a **pointer variable** (or **pointer**), for this purpose.

There are four kinds of dynamic data structures use pointers in Pascal. They are : **linked lists**, **stacks**, **queues**, and **trees**. A **linked list** or simply **list** is a sequence of nodes in which each node is linked or connected to the node following it. Each node in the list has two fields : the first field contains data and the second field is a pointer to the next list element.

A stack can be thought of as a linked list in which each new node is inserted at the head of the list and each deletion removes the current head of the list. Inserting a node is a **push** operation and deleting a node is **popping** the stack.

A queue is a linked list used to model things such as a line of customers waiting at a checkout counter or a stream of jobs

waiting to be printed by a line printer. In a queue, all insertions are done at one end (the rear of the queue) and all deletions are made from the other end (the front of the queue).

So far we have involved list elements or nodes with a single pointer field. It is possible to have lists of elements with more than one link. We call it a multiply-linked list.

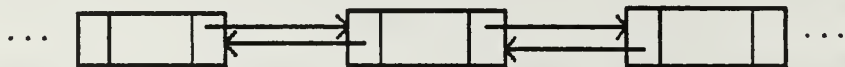


Figure 2-2 A Sample of Multiply-linked List

A special kind of multiply-linked list that has wide applicability in computer science is a data structure called a **binary tree**. (See Figure 2-3)

The details of binary tree and traverse or search of a binary tree are subjects of Data Structure, to go further into them please reference Data Structure text books.

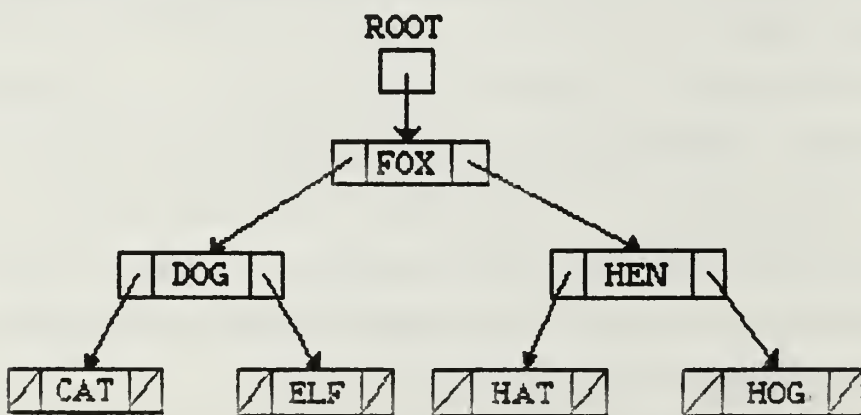


Figure 2-3 A Sample Tree

D. SUMMARY

Pascal's primary goal was to be a good language for teaching

programming. This led to subsidiary goals for reliability, simplicity, and efficiency. Pascal has been very successful in these areas.

Pascal is a particular suitable language with which to learn programming. Most modern programming concepts are available in Pascal. In sharp contrast to BASIC, things become much more complex as programs grow in size. Pascal programs, by contrast, expand gracefully. The concepts we learn with Pascal are applicable in almost any programming environment.

Much of the criticism Pascal has received results from trying to use it for purposes for which it was not designed. For example, Pascal has been criticized for its lack of a separate compilation facility, even though such a facility is not especially important in teaching programming (the language's intended application). Indeed, it is to Pascal's credit that it has been so successfully applied in so many areas for which it was not intended.

III. THE PROGRAMMING PRINCIPLE OF dBASE III PLUS

A. AN OVERVIEW OF dBASE III PLUS

A database is a central repository of related information. To paraphrase this, a database is a physical grouping of a collection of individual, but related, bits and pieces of information. The difficulty in building an effective database system is not in the mechanics of construction, but in the intelligent design and planned use of the database.

As an example, if one wants to maintain information about each and every individual employed in an organization, it is necessary to create a base of data about all the employees. This base of data could contain, for example, information about each employer's employee-number, name, salary, year of hire, and date of last promotion. This base will subsequently provide immediate access to the type of information sought. Database can and are being maintained for every subject from astronomy to zoology. Computers, because of their speed and accuracy, are the information processor, the physical means, of creating and subsequently accessing these databases.

dBASEIII PLUS is defined as a relational database manager, that is, this software helps create and maintain a relational database. A relational database is one in which the data is arranged in the form of a matrix, with the rows of the matrix forming each individual record in the database, and the columns of the matrix forming the individual fields of information. Using such a database, one can establish a relationship between two or more databases, by using a common key field of information.

B. THE dBASE III PLUS SYSTEM

So where does dBASE III PLUS fit in with all of the previous concepts and definitions?

dBASE III PLUS is the name of a software package marketed by Ashton Tate, Inc., of Culvery City, California; it is a very powerful tool for the development of microcomputer business applications. dBASE III PLUS is a data manager. It is a piece of software that lets the user have full freedom in the conceptualization and creation of database for all types of business applications. Since business depends on timely information dissemination, the value of a powerful, programmable utility for database generation, maintenance, and query cannot be overstated. dBASE III PLUS is defined as a relational database manager, that is, this software helps create and maintain a relational database. dBASE III PLUS can be executed on a variety of microcomputers, under any one of the popular operating systems.

C. DATA TYPES OF dBASE III PLUS

Data is defined as something known or assumed; facts from which a conclusion can be inferred. Data usually represents some aspect of the physical world around us, such as a list of names and addresses, the temperature of the room, today's date and time, or a bank statement.

A data type is a high level representation of data as seen by the user which has a corresponding binary form understood by the computer. Data types allow people to write programs using data representations with which they are comfortable. The high level representation is maintained internally as a binary format processed by the computer.

Each language provides a limited number of elementary data

types. Complex data structures can be constructed from the elementary types. dBASE III PLUS provides : **Character**, **Numeric**, **Logical**, **Date**, and **Memo**. Using these data types, complex data structures useful in representing a multitude of real world situations can be constructed.

1. Character Data Type

A field defined as a **character field** accepts any character of data entered. Character data is used to represent letters of the alphabet, numbers, and special characters. In dBASE III PLUS, the character type is made up of the set of all ASCII characters. A **character string**, often just called a **string**, is any sequence of ASCII symbols. When a number is represented as a character type, it must first be converted to numeric data before calculations can be performed with it. It is often convenient to use the character data type for numbers such as telephone numbers, addresses, and inventory stock numbers which will not be used in calculations. Some of the more common operations performed on strings are:

- Concatenating strings (linking them together)

- Splitting up strings into "substrings"

- Testing strings for equality

- Finding substrings (string patterns)

The character variable contains textlike information: "David Smith", "1234 Fifth Street", "Computer Science Department". Any information between the quote signs will be taken as a character string. The maximum length of a string is 254 characters. The minimum length of a string is 0.

Because dBASE III PLUS is a business oriented language, it deals much more in text manipulation than computer languages such as BASIC and Pascal.

2. Numeric Data Type

A field defined as a **numeric field** will only accept the digits 0 through 9, the decimal point, and the negative sign(-) as data. (Trying to force character data into a numeric field will lock up the keyboard.) Numeric data is used to represent integers or decimal quantities that will undergo computations. dBASE III PLUS allows a wide range of numbers and has adequate precision for most business and scientific applications. It maintains an internal precision of fifteen or sixteen digits, depending on the size of the number. This allows dBASE III PLUS to be accurate on calculations with fairly large numbers without round off error. Internally dBASE III PLUS represents numbers with the IEEE long real (64-bit) binary floating point representation. A binary floating point representation is the computer's equivalent to scientific notation. Each number contains three parts: the **sign**, either + or -, the **significand** which represents the significant digits of the number, and the **exponent** which multiplies the significand by the appropriate power to yield the correct binary point position in the final result.

3. Logical Data Type

A logical data field is one which is of a predefined length, 1 character, and will accept as input either the letters T or Y (for TRUE/YES) or the letters F or N (for FALSE/NO). The actual data is stored exactly as entered, but will be displayed on the screen or printer as .T. or .F. only. If no data is entered, the default is .F.

Logical data fields are used to represent types of data when there are only two choices for any element, such as male/female, positive/negative, yes/no, dead/alive. dBASE III PLUS can perform

conditional tests which depend on the value of a logical field.

4. Date Data Type

A date data field is also of a predefined length, eight characters, and dBASE III PLUS presumes that you will be subsequently entering a date of the format MM/DD/YY. At the time of actual data entry into this field, dBASE III PLUS automatically checks for the accuracy of the data entered. For example, an entry of 12/35/85 would invoke a beep and an error message. The built-in edit even checks for a leap-year! Date fields are very useful in that they reduce the amount of programming effort needed for routines computing time lapse, since you can add numbers to or subtract numbers from, date fields, or you can add or subtract two date fields directly.

5. Memo Data Type

A memo data field is also of a predefined length, 10 characters in the file itself, and automatically contains the word **memo** for data. Through the use of this field, you can maintain memos for individual records. Each memo could be up to 4000 characters long if the built-in dBASE III PLUS word processor is used, or can be any length if it is set up with a commercial word processor. dBASE III PLUS makes use of an external file in which it stores the contents of the individual memos, and hence the memo can have the capacities mentioned above. This external file will have the same primary name as the dBASE III PLUS file, but will have the .DBT extension for the secondary name. dBASE III PLUS maintains this file in an internally usable form.

D. DATA STRUCTURES OF dBASE III PLUS

dBASE III PLUS contains powerful high level commands which allow people to create and manipulate sophisticated data structures easily. There are several common models used to represent data structures. dBASE III PLUS uses the relational model: the data is represented in flat (two-dimensional) tables composed of rows and columns. In relational databases, a two-dimensional table is known as a "relation", and operations on these tables can be described with mathematical precision. However, the relational model is very powerful and all other common data models can be represented using two-dimensional (relational) tables.

The CREATE command is used to create a data file structure matching the body of data that it is intended to represent. This structure can store data as individual records and the data can then be easily accessed and manipulated.

1. The Data File Structure

The dBASE III PLUS data file can be seen as a two-dimensional table containing the following properties:

- a. All items in a column are of the same data type; that is, the data file is column-homogenous. A column in dBASE III PLUS is a field.
- b. Each column (or field) must have a distinct fieldname. No duplicate fieldnames are allowed.
- c. Each row is a record and is assigned a number. The record number assigned is relative to the record's position in the data file.

A conceptual view of a dBASE III PLUS data file is:

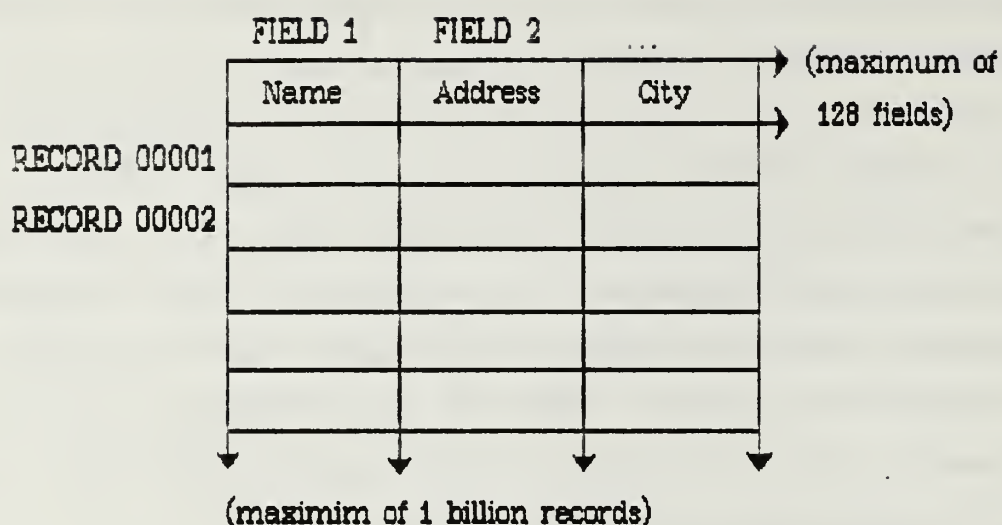


Figure 3-1

The limitations on dBASE III PLUS data files are:

File limits (maximum sizes)

Number of records/file	1 billion
Record size (bytes)	4000 bytes in .DBF file
Number of fields/record	128

Field limits

Character fields	254 bytes
Logical fields	1 byte
Numerical fields	19 bytes
Date fields	8 bytes
Memo fields	10 bytes in .DBF file

.DBF file size limit only by operating system, hardware or word processor used.

2. Indexing

One of the most important features of any database system is the ability to find a specific data item from among many item

quickly. Search by index is the best way to shorten long sequential search times to only a couple of seconds.

When the INDEX ON <key> command is issued in dBASE III PLUS, a separate index file is created based on the key expression specified in the command line. The followings are examples of creating indexed files:

- . USE Customer (Select data file)
- . INDEX ON Names TO Customer (Generate index)

This index allows rapid searches of the **Customer** database by **Name**. Any record can be found in two seconds or less with the FIND or SEEK command.

E. PROGRAMMING IN dBASE III PLUS

In dBASE III PLUS, a program is actually a set of commands in sequence. To create a program, at dot-prompt, just type in:

```
.CREATE COMMAND pgm <CR>
```

This command informs dBASE III PLUS of your intention to create a **command file** (program) called pgm.PRG. The screen is then erased; the dBASE III PLUS word processor takes over; there is no dot-prompt; and whatever you key in will remain on the screen, until you either SAVE it(Ctrl-W), or DELETE it(Ctrl-Q).

To invoke the execution of a program, simply request dBASE III PLUS to:

```
.DO PGM <CR>
```

To make changes to an existing program, use the following command:

```
.MODIFY COMM pgm <CR>
```

1. Parameter Passing

The main value of parameter passing occurs when a low

command file can be called from many different programs in many situations and must be free of the naming convention used in the calling program.

There are two kinds of variables, PUBLIC(global) and PRIVATE(local). A PUBLIC variable can only be released by the programmer while a PRIVATE variable is released by dBASE III PLUS when the program returns from the command file in which the variable was created.

All variables created in a command file are PRIVATE unless specifically declared PUBLIC. All variables created in higher level command file are available to the lower level command files unless there is a clash. If a variable is declared PRIVATE in a command file and there exists another variable from a higher level with the same name, then dBASE III PLUS has to decide which variable is to be referenced and which is to be hidden. Until the PRIVATE variable is released, dBASE III PLUS hides the higher level variable and all references of the variable name refer to the lower level variable.

The variable status, PUBLIC or PRIVATE, can be directly specified:

PUBLIC name,city

PRIVATE salary

Any variable can be released with the RELEASE command:

RELEASE name,city

dBASE III PLUS automatically releases a PRIVATE variable when the user leaves a command file with RETURN, CANCEL, or QUIT. Usually variables are passed from one command file to another without specific instructions. With parameter passing, the high level command file does not have to follow the same naming convention that the low level command file uses.

PUBLIC variables should be used sparingly. Generally variables should not be made PUBLIC without some specific reason. While debugging, the most important variables should be PUBLIC so that they can be inspected if the program crashes and control returns to the dot prompt (thereby releasing all the private variables).

2. Control Transfers From One Program To Another

Just as you invoke the execution of a program by asking dBASE III PLUS to DO <program name>, you can invoke the execution of another program from within the first one in the same way. The **calling** program, at some logical point in its execution, transfers control to the **called** program. At the end of the execution of the **called** program, control is automatically transferred to the instruction after the DO instruction that passed control to the **called** program.

This concept of transferring control to subprograms and then receiving control back at the main program is very important to the programmer, since it permits the breakdown of a large complicated system into subset of logically connected, more manageable subprograms. This makes the system much more comprehensible not only to other programmers but also to the creator of the system.

F. HOW DOES dBASE III PLUS IMPLEMENT THE RELATIONAL MODEL

In dBASE III PLUS, there is a very powerful command:

SET RELATION TO

The SET RELATION creates a link between two data files. Its power stems from the fact that dBASE III PLUS will automatically look up related information from another file. This means that

FORM commands, will perform a search of another database.

As an example, suppose sales information is kept in one file and time of shipment information in a second file. SET RELATION will allow the user to produce reports on the transactions in which dBASE III PLUS looks up the sales information at each transaction. For example, in S_INQUERY.PRG:

```
SELECT 1
```

```
USE b:s_contra INDEX b:s_conind
```

```
SELECT 2
```

```
USE b:tmofship INDEX b:tmshipdx
```

then in SALELIST.PRG:

```
SET RELATION TO snumber INTO tmofship
```

this will create relationship between **s_contra** and **tmofship** with **snumber** as the common key.

SET RELATION is equivalent to a user who always performs a SEEK command at each record but it is faster than writing the individual commands and it enables many dBASE III PLUS command to utilize the second file. This gives multi-file capabilities to the nonprogrammer.

G. I/O PROCESSING OF dBASE III PLUS

The I/O processing of dBASE III PLUS is more concerning about the communication between the programmer and operator. This communication can be broken down into two categories:

1. Output: The programmer talks to the operator.
2. Input: The operator talkks to the programmer.

As mentioned above, dBASE III PLUS was designed for microcomputers. Most current I/O devices used on microcomputers are CRTs(or screen). Therefore, dBASE III PLUS contains very powerful screen handling capabilities discussed below.

powerful screen handling capabilities discussed below.

1. Output

In dBASE III PLUS, output screen handling refers to the process by which the programmer talks to the operator. The output commands used to support screen handling can be categorized according to the mode in which they work:

a. FORMATTED MODE

@ <coordinate> SAY

This command places its output at the screen location specified by the programmer, thus formatting the screen.

In addition, it has options which allow the programmer to modify the display of its data, thus formatting its output.

b. UNFORMATTED MODE

General:

?

??

Specialized:

DIR

DIRECTORY

DISPLAY

LABEL

TYPE

These commands are dependent upon the current cursor position and begin their output at that location. The most frequently used commands in communicating with the operator in dBASE III PLUS programming applications is the @...SAY command because of the degree of control it offers. The ? command is

when a screen scrolling effect is desired.

TEXT ... ENDTEXT is a structured output command, rather than a structured programming command because it has no effect at all on program flow. It is simply a convenient way of outputting large amounts of unformatted text. The literal text must be contained within the TEXT ... ENDTEXT structure, and therefore is a constant in the command file.

c. FORMATTED SCREEN

The <coordinates> specified in the @ ... SAY command control where the output will appear on the screen. The syntax and range for computers with 24X80 screens is:

```
@ <coordinates> SAY <expression>
<coordinates> = <row>,<column>
<row> = numeric expression, range 0-23 (line)
<column> = numeric expression, range 0-79
```

Note that minus numbers cannot be used with this relative addressing operator.

d. FORMATTED OUTPUT

In addition to formatting the screen, we can also format the individual picture of each data item when we display it. The @...SAY command offers the programmers a variety of options for displaying data in a format different than the format in which it exists. For example, a numeric field cannot contain commas, but it can be displayed with commas when output with @...SAY command. The syntax and formatting options are:

```
@ <coordinates> SAY <expression> <format option>
<format option> = PICTURE '<picture template>'
                  | FUNCTION '{<function>}'
```

| FUNCTION '{<function>}'
| USING '{<using symbol>}'

2. Input

Input screen handling refers to the process by which the operator talks to the programmer. Output screen handling is the reverse of this. Operator input must be carefully handled. This is the time to trap all the possible errors so that the data in the database is always known to be accurate and good. The input commands used in dBASE III PLUS can be categorized according to the mode in which they work.

a. FULL-SCREEN MODE

@...GET

This command places a variable (field or memory variable) at the screen location specified by the programmer. In addition, it has options which allow the programmer to restrict the operator's input.

READ [NOUPDATE]

This command places the cursor in variables which have been placed on the screen with @...GET. This enables the operators to enter or edit data in the variable.

b. COMMAND-LINE MODE

Memvar:

ACCEPT (character type)

INPUT (date, logical, and numeric types)

WAIT (character type, one character only)

Of course, the most frequently used commands for receiving communications from the operator is the @...GET/READ

often used to simply pause the program execution until the operator hits any key; the keystroke itself is usually disregarded.

ACCEPT and INPUT are usually used for quick utility type applications where a high degree of error trapping is not required, or when it is desirable to give the operator lots of flexibility, such as in programmer's utilities. ACCEPT will only accept a character type literal while INPUT will accept an expression of any data type.

Screen placement and appearance of @...GET are the same as for @...SAY, which are mentioned above. There is one combination form, the @...SAY...GET which places the GET <variable> on the screen immediately following the SAY <prompt>.

```
@ 5, 0 SAY 'Your name:'
```

```
@ 5,14 GET memvar
```

```
@ 5, 0 SAY 'Your name:' GET memvar
```

These both produce the same results. The first form makes writing some screen easier. The second form runs faster.

c. FORMATTED INPUT

The @...GET command offers the programmer a variety of options for limiting the data that the operator can enter. For example, a character type variable can be limited to accepting only numbers from the keyboard. The syntax and formatting options are:

```
@ <coordinates> GET <variable name> <format option>
```

```
<variable name> = A currently active memvar
```

```
| A field in the currently selected  
database file
```

```
<format option> = PICTURE '<picture template>'
```

```
| FUNCTION '{<function>}'
```

| RANGE<n1>,<n2>

d. FORMAT FILE

Format files are like command files except that they contain only @...SAY and @...GET commands and comments. Format files allow the formatting of the screen during the full-screen interactive database commands APPEND, CHANGE, EDIT, and INSERT. An open format file also affects the execution of the READ command by clearing the entire screen, resetting the GET counter, and redisplaying its SAYs and GETs. DBASE III PLUS can have one format file for each of its ten work areas if this will not exceed the limit of thirteen simultaneously open files of all kinds. It also closes any open format file in the currently selected work area when a new database file is opened or any current one closed.

This command open a format file:

SET FORMAT TO <format filename>

These commands close all open format file:

CLEAR ALL

CLOSE DATABASES

These commands close only the open format file in the currently selected work area:

CLOSE FORMAT

SET FORMAT TO

USE

H. SUMMARY

In dBASE III PLUS, there are no ARRAYs, RECORDs, SETs, FUNCTIONs, ENUMERATED TYPEs nor LINKED LISTs. These were considered to be necessary for structured programming, but quite an overhead to a database system. dBASE III PLUS uses nothing

but two-dimensional tables to implement its data structures and file organizations. dBASE III PLUS consists of a set of commands each with a "syntax." Each command is extremely flexible and can perform an infinite number of variations on a particular task. It's invaluable flexibility requires more learning effort on the part of the user.

The control structures such as SEQUENCE, BRANCH, REPETITION and those algorithms which are used in most programming languages are available in dBASE III PLUS too. But dBASE III PLUS does not support recursion.

In general, dBASE III PLUS is a versatile database manager. It is designed to handle the many business applications in which the user needs to manage large amounts of repetitive information. It can function as a simple file manager. It can handle the complex issues in relating files to one another, and it can be used as a complete language like Pascal or BASIC.

IV. STUDY AND ANALYSIS OF ORIGINAL PROGRAM

A. BACKGROUND

The original program - written in Pascal, is a typical inventory management system software, written for Import/Export Company, which purchases feed stuffs such as hays, grass, etc., from farmers in California. Those feed stuffs are compressed and packed into 9X40 foot containers and shipped to foreign countries. This program was written in order to keep track of **the sales and the purchase**. For instance, how many tons of the grass have been purchased? How many of them have been shipped? At what price, and what time? How many tons left need to be shipped to complete a contract? The analysis of the algorithm and data structures used in this program are discussed in detail in the following sections.

B. THE ENTITY-RELATIONAL DIAGRAM

Sales and Purchases are an independent process within Hanaoka, as can be seen on the next page (Figure 4-1). The E-R diagram of Hanaoka was divided into two sub-diagrams. The upper part is for Purchases and the lower part is for Sales. There are some identical components in these two sub-diagrams.

In Sales, each sale record contains one customer and several time-of-shippings records. The sale record is used to update the sale contract according to the contract number. The sale contract uses the invoice number (input from screen) concatenated with the contract number as a new invoice number to update the sales shipment record. Data is passed from the sales record. Also the sales record can be used to check the hash file for the duplicate

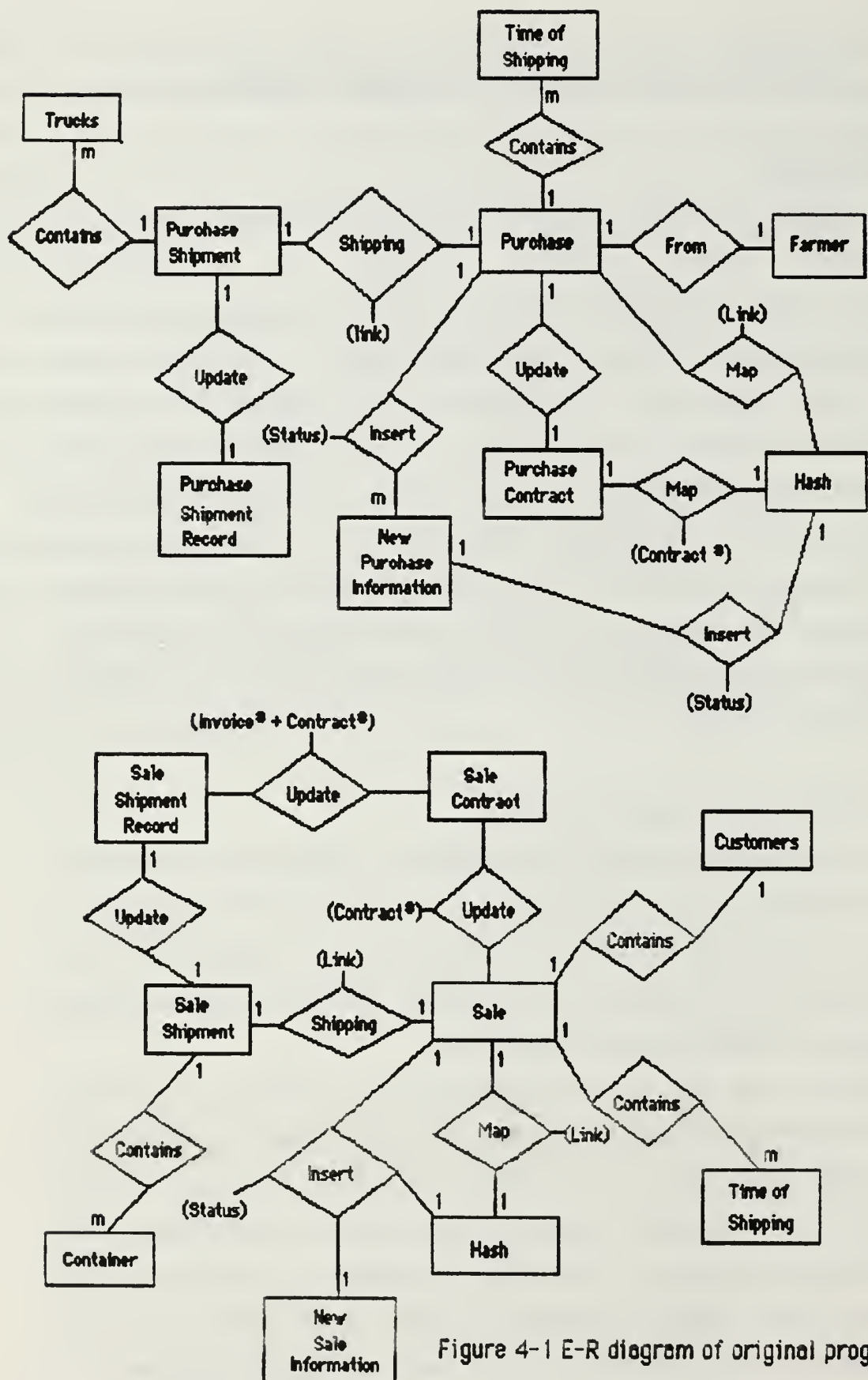


Figure 4-1 E-R diagram of original program

key. If there is no duplicate key then this sales record can be inserted into an empty slot of the hash file according to the status (empty or occupied) of that slot. Each sales shipment can be updated by the sales shipment record and each sales shipment contains several containers

In Purchase, each purchase record corresponds to each farmer. Each purchase record can contain several time-of-shipping records. New purchase information can be stored into the hash file and purchase file by checking the slot's status (empty or occupied). One purchase needs one purchase shipment record which can contain several trucks records, and can be used to update purchase shipment records.

C. PROGRAM STRUCTURES

1. Sales

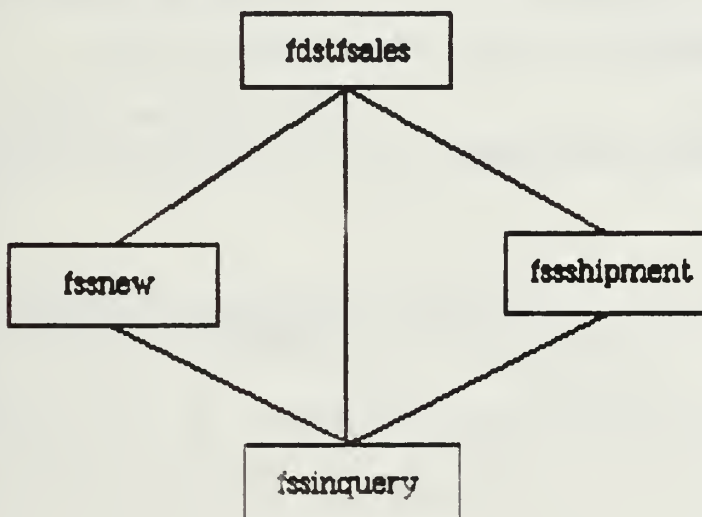


Figure 4-2 Structured diagram of Sales

The followings is the program's outline for Sales:

(For details, please reference APPENDIX A)

```
*****  
*                               MAIN PROGRAM                               *  
*****
```

```
PROCEDURE fdstfsales;
{ procedure to handle all operations concerning feed stuff sales }
BEGIN
END;
```

```
*****
*                                     FSSNEW                                     *
*****
```

```
PROCEDURE tosinfo;
{ handles one input array timeofship, different proc since the
input format differs from other input, i.e., makes procedure
READNEXTINPUT too long }
BEGIN
END;
```

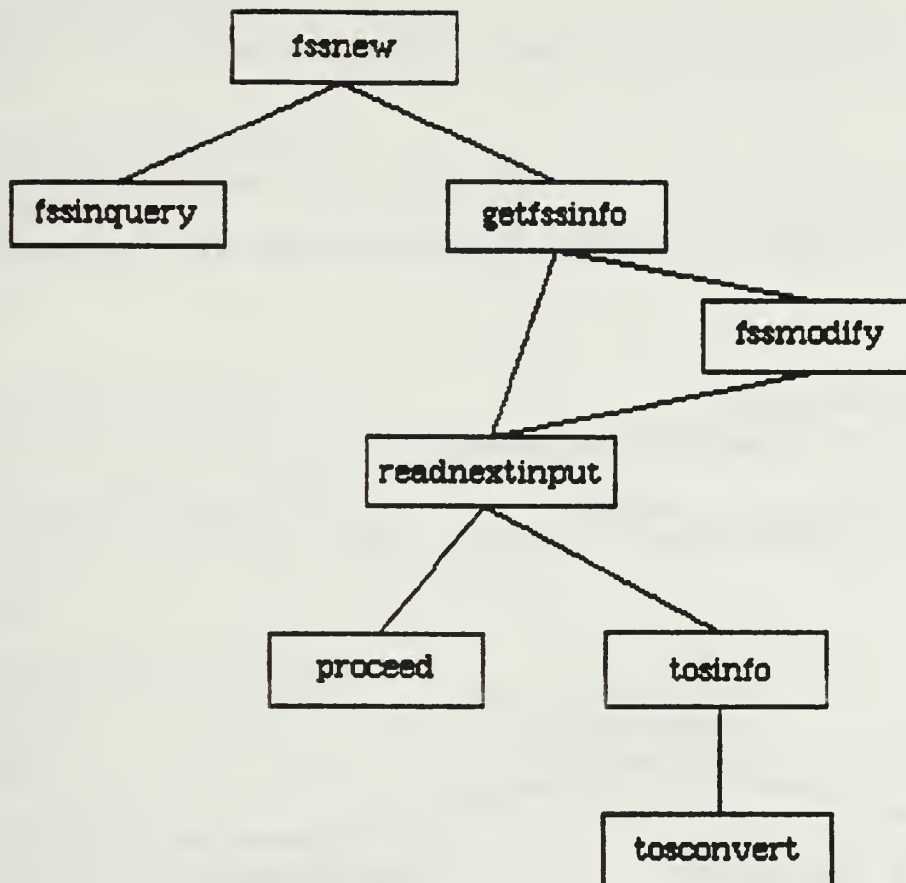


Figure 4-3 Structure diagram of FSSNEW

```

PROCEDURE readnextinput;
{ handle one input at a time, var lineno determines which
input }
BEGIN
END;

```

```
PROCEDURE fssmodify;
{ re-reads any specified input once more }
BEGIN
END;
```

BEGIN
END;

BEGIN
END.

```
*****
*                                     *
*                                     FSSSHIPMENT                                     *
*                                     *
*****
```

```
SEGMENT PROCEDURE fssshipment;
```

```
  PROCEDURE contconvert;  
  { converts container array input from string to appropriate data  
    type }  
  BEGIN  
  END;
```

```
  PROCEDURE computeart;  
  { does all necessary conversion (lbs->shorttons) and computation  
    }  
  BEGIN  
  END;
```

```
  PROCEDURE getshipinfo;  
  { get all pertinent info for a shipment}
```

```
  FUNCTION sproceed : boolean;  
  { same as FUNC proceed }  
  BEGIN  
  END;
```

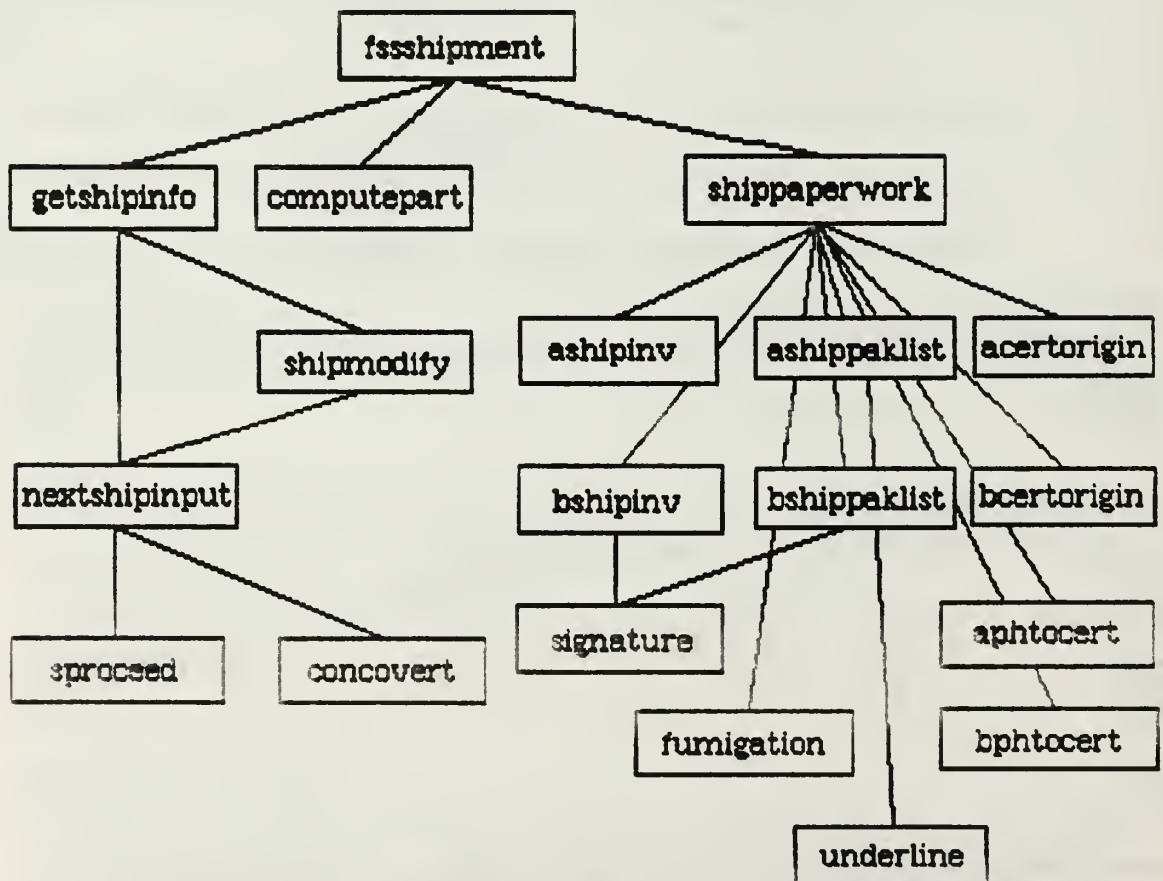


Figure 4-4 Structure diagram of FSSSHIPMENT

```

PROCEDURE nextshipinput;
{ reads one input at a time }
BEGIN
END;

PROCEDURE shipmodify;
{ re reads specified input once more }
BEGIN
END;

BEGIN
END;

PROCEDURE shippaperwork;
{ produces five documents pertaining one shipment }

    PROCEDURE signature;
    { write closing (like rubber stamp) }
    BEGIN
    END;

    PROCEDURE underline;
    { prints "-" for specified no of times }
    BEGIN
    END;

    PROCEDURE ashipinv;
    { top half of invoice }
    BEGIN
    END;

    PROCEDURE bshipinv;
    { bottom half of invoice }
    BEGIN
    END;

    PROCEDURE ashippaklist;
    { top half of packing list }
    BEGIN
    END;

    PROCEDURE bshippaklist;
    { bottom half of packing list }
    BEGIN
    END;

    PROCEDURE acertorigin;
    { top half of cert of origin }
    BEGIN
    END;

    PROCEDURE bcertorigin;

```

```

    { bottom half of cert of origin }
BEGIN
END;

PROCEDURE aphytocert;
{ top half of phytosanitary cert }
BEGIN
END;

PROCEDURE bphytocert;
{ bottom half of phytosanitary cert }
BEGIN
END;

PROCEDURE fumigation;
{ fumigation certificate }
BEGIN
END;
BEGIN { shippaperwork }
END;
BEGIN
END.

```

```

*****
*                                     *
*                               FSSINQUERY                               *
*                                     *
*****

```

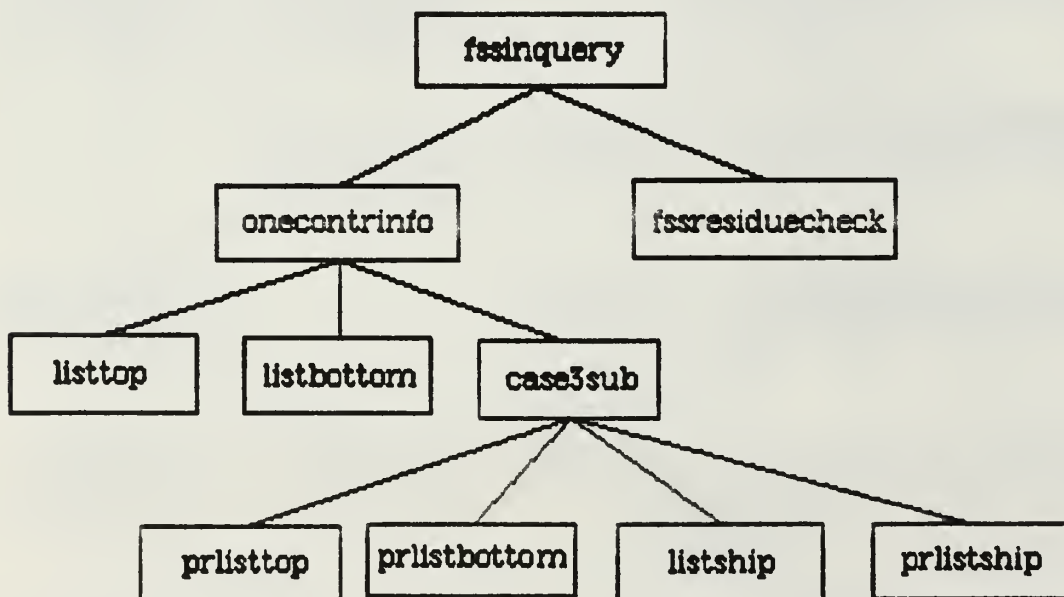


Figure 4-5 Structure diagram of FSSINQUERY

```

SEGMENT PROCEDURE fssinquery;
{ handles queries of following types:

```

1. list all customers by company name.
2. list all contracts of one customer.
3. list all information (include all shipments made) of one contract
4. list available spaces in FSSFILE and FSSSHIPFILE }

```
PROCEDURE listtop;  
{ list top half of contract information }  
BEGIN  
END;
```

```
PROCEDURE listbottom;  
{ list bottom half of contract information }  
BEGIN  
END;
```

```
PROCEDURE prlisttop;  
{ same as listtop but outputs to printer }  
BEGIN  
END;
```

```
PROCEDURE prlistbottom;  
BEGIN  
END;
```

```
PROCEDURE listship;  
{ list shipment info to console }  
BEGIN  
END;
```

```
PROCEDURE prlistship;  
{ list shipment info to printer }  
BEGIN  
END;
```

```
PROCEDURE onecontrinfo;  
{ handles query of type 3 }
```

```
    PROCEDURE case3sub;  
        { handles the shipment info of one contract and printout of  
          contract info to the printer }  
    BEGIN  
    END;  
BEGIN  
END;
```

```

PROCEDURE residuecheck;
{ handles query of type 4 }
BEGIN
END;
BEGIN { FSSINQUERY }
END; { Note query 1 & 2 is handled in this main procedure }

```

2. Purchase

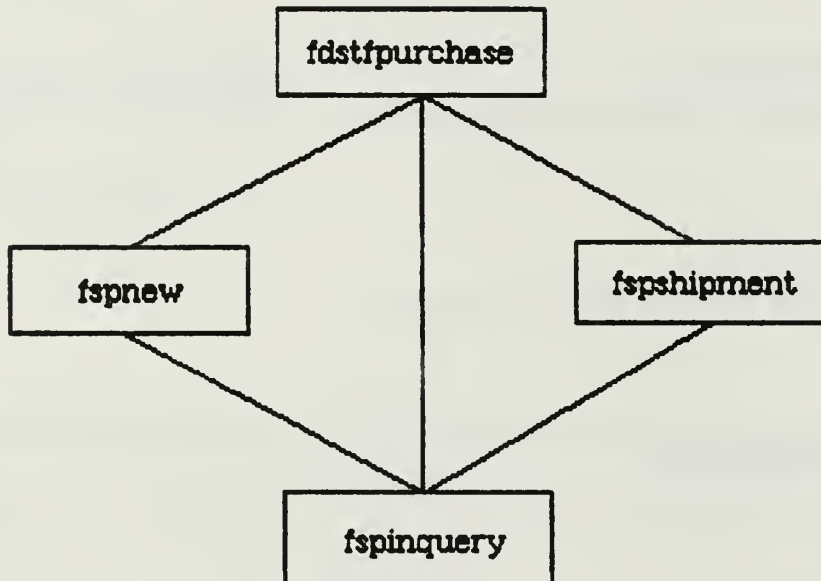


Figure 4-6 Structure diagram of Purchase

Followings are the program's outline of purchase:
 (For details, please reference APPENDIX A)

===== PURCHASE =====

```

*****
*                                     *
*                               MAIN PROGRAM                               *
*                                     *
*****

```

```

SEGMENT PROCEDURE fspinquiry;
{ to prompt the query concerning the feed stuff purchase file }
BEGIN
END;

```

```

SEGMENT PROCEDURE fspnew;
{to input new purchase contract into the files }
BEGIN
END;

```

```

SEGMENT PROCEDURE fspshipment;
{ to input shipment information for the existing contract }
BEGIN
END;

PROCEDURE fdstfpurchase;
{ procedure to handle all operations concerning feed stuff purchase }
}
BEGIN
END;

```

```

*****
*                                     *
*                               FSPNEW                               *
*                                     *
*****

```

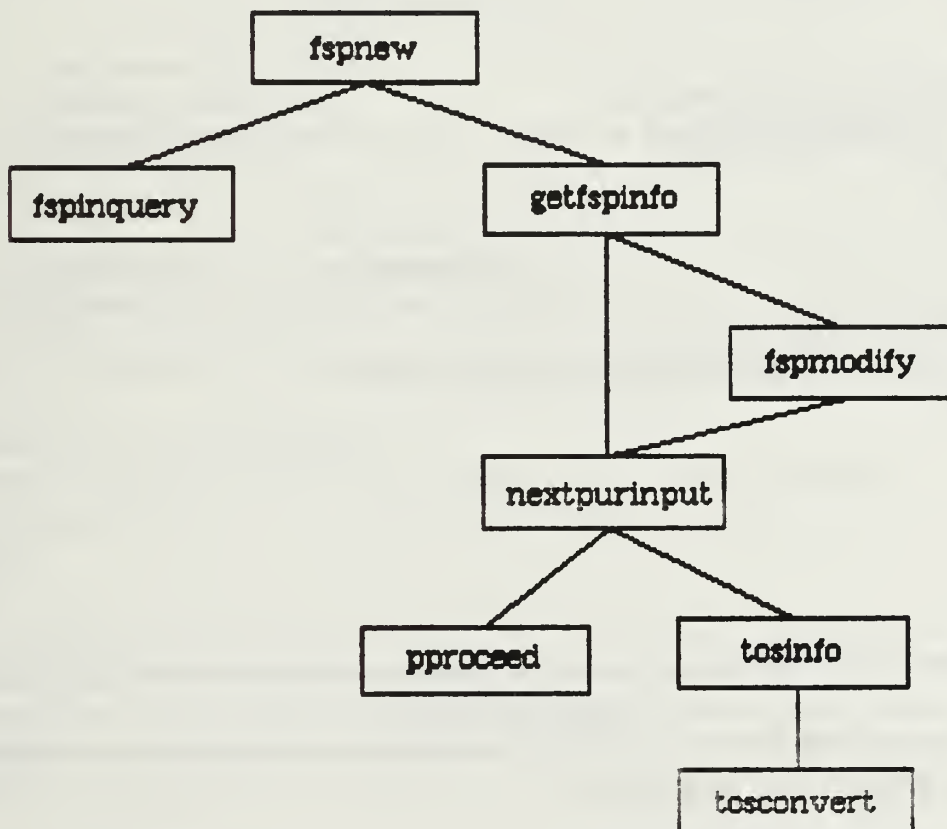


Figure 4-7 Structure diagram of FSPNEW

```

SEGMENT PROCEDURE fspnew;

PROCEDURE tosconvert;
{ converts timeofship array input }

```

```

BEGIN
END;

PROCEDURE getfspinfo;
{ input all pertinent info for new purchase contract }

    FUNCTION pproceed : boolean;
    { return true if the input line is not empty, so not to allow
      null input }
    BEGIN
    END;

PROCEDURE tosinfo;
{ handles one input array timeofship, different proc since the
  input format differs from other input,i.e., makes procedure
  readnextinput too long }
BEGIN
END;

PROCEDURE nextpurinput;
{ handles one input at a time, var lineno determines which
  input }
BEGIN
END;

PROCEDURE fspmodify;
{ re reads any specified input once more }
BEGIN
END;
BEGIN
END;
BEGIN
END;

*****
*                                     FSPSHIPMENT                               *
*****
SEGMENT PROCEDURE fspshipment;

PROCEDURE truckconvert;
{ converts truck array input from string to appropriate data type
  }
BEGIN
END;

```

```

PROCEDURE gettruckrate;
{ decides the price rate and convert into appropriate data type }
BEGIN
END;

```

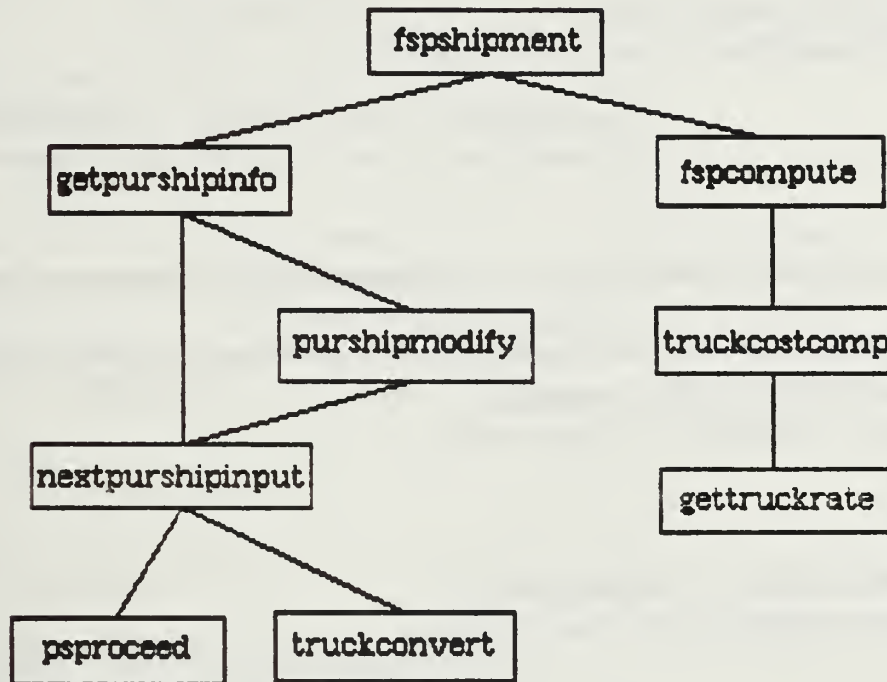


Figure 4-8 Structure diagram of FSPSHIPMENT

```

PROCEDURE truckcostcomp;
{ decide the rate base (lbs or short tons) and does the
  computation }
BEGIN
END;

```

```

PROCEDURE fspcompute;
{ compute the numeric data according the rates and bases }
BEGIN
END;

```

```

PROCEDURE getpurshipinfo;
{ get all pertinent info for a shipment }

```

```

FUNCTION psproceed;
{ same as pproceed }
BEGIN

```

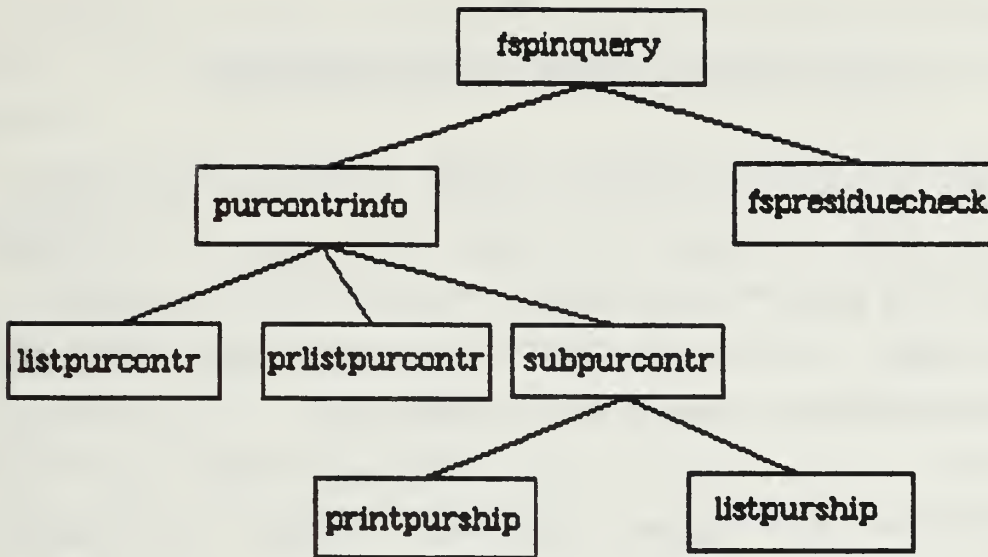



Figure 4-9 Structure diagram of FSPINQUERY

```

SEGMENT PROCEDURE fspresiduecheck;
{ handles query of type 4 }
BEGIN
END;

```

```

SEGMENT PROCEDURE fspinquery;
{ handles queries of following types:
  1. list all commodities.
  2. list all contracts of one commodities.
  3. list all information (include all shipments made) of one
     contract
  4. list available spaces in fspfile and fspshipfile }
BEGIN
END;

```

V. ANALYSIS AND DESIGN OF NEW PROGRAM

In order to use the dBASE III PLUS to rewrite the original program, we need to replace the existing set of files by its logical equivalent. The term " file structure, " refers to the complete set of stored data, its division into component sets, and the relationships that exist among those components. In most cases, the component sets are files or repeating groups within files. The relationships that exist among components include access keys, ordering techniques, redundancies, pointers, and so forth.

It is the whole file structure that must be logicalized. To talk about deriving logical equivalents of each current file is fruitless, because the very decision that some of these files should exist at all may have been physical. So we have to go back and think about the whole (the union of all data elements), and start from scratch to divide those files into its component pieces.

The process of dividing those files into its components is called **normalization**, which is a process to analyze the functional and multivalued data dependencies. The objective of normalization is to avoid redundancy and update, insert, and deletion anomalies. Each time we normalize a file, we replace a multipurpose file by two or more files. These files together can accomplish the same set of purposes as the original. The new files are always more singleminded than the one they replace. It frequently happens that the new equivalent files need to be normalized again.

There are different levels of normalization, named **1st normal form**, **2nd normal form**, **3rd normal form**, **BC normal form**, **4th normal form**, **5th normal form** and **domain/key normal form**.

A relation is in 1NF (1st normal form) if it contains no repeating groups. It is a simple matter to produce 1NF from an unnormalized relation.

A relation is in 2NF if it is in 1NF and there are no partial dependencies.

A relation is in 3NF if it is in 2NF and there are no transitive dependencies. For most practical databases, 3NF is sufficient.

A relation is in BCNF (Boyce Codd) if it is in 3NF and every determinant is a candidate key.

A relation is in 4NF if it is in BCNF and there are no multivalued dependencies.

Before rewriting the original program, we have to normalize the relations of the files used in the original program, then use those normalized logical file structures to create new Entity-Relational diagrams and a new relational database written with dBASE III PLUS.

A. NORMALIZATION OF SALES

As we can see in Appendix A, the file structures declared in the original program are listed as follows:

```
FSSFILE { Status, Number, ContrDate, Customer.Name,  
          Customer.Addr, Customer.ContrNo, Commodity, Pricebase,  
          Lc.Number, Lc.ExpDate, Lc.ShipDate, Lc.Bal, Lc.Amount,  
          TimeOfShip, TotalShip, BalOfShip, IssueBank, DrawBank,  
          MitiNo, NofShipment, ShipmentInfo}
```

```
TimeOfShip { Month, Wgt, Bal, UnitPrice }
```

```
FSSHASHFILE { Status, Number, Name, Commodity, Link}
```

```
FSSSHIPFILE { Status, InvoiceNo, Name, Origin, Dest, Etd,  
              InvoiceDate, TotalBales, TotalNet, Container,  
              NofCont}
```

```
Container { Number, Bales, Net }
```

FSSCONTRACT { same as FSSFILE }

FSSDUMMY { same as FSSFILE }

FSSSHIPREC { same as FSSSHIPFILE }

1. 1st Normal Form

There are three file structures used in **Sales**, with two repeating groups in FSSFILE (as indicated with underline) and one repeating group in FSSSHIPFILE. One temporary file FSSHASHFILE is used as an intermediate file to map into FSSFILE for updating or inserting records. This is not necessary in the relational database.

The field STATUS in FSSFILE and FSSSHIPFILE is used to keep track of the space allocation. In dBASE III PLUS, the APPEND and INSERT take care of this. Again we can eliminate these from the files. The field SHIPMENTINFO is used as a pointer to point to the shipment information. Actually it is used to create the relationship between FSSFILE and FSSSHIPMENT. Again we can eliminate this, because in the relational database, a relation can be created by using the same primary key.

To put these files in 1NF, the repeating groups must be removed from within these files. After 1st normalization, we have these new relations:

FSSFILE { Number, ContrDate, Customer.Name, Customer.Addr,
Customer.ContrNo, Commodity, Pricebase, Lc.Number,
Lc.ExpDate, Lc.ShipDate, Lc.Bal, Lc.Amount, TotalShip,
BalOfShip, IssueBank, DrawBank, MitiNo, NofShipment }

FSSSHIPFILE { InvoiceNo, Name, Origin, Dest, Etd, InvoiceDate,
TotalBales, TotalNet, NofCont }

CONTAINER { Number, Bales, Net }

TIMEOFSHIP { Month, Wgt, Bal, UnitPrice }

2. 2nd Normal Form

We have separated repeating groups from files, and we need to create relations between these new files. In TIMEOFSHIP, the field MONTH could be a primary key, but the month can be repeated every year. So, we can compose NUMBER from FSSFILE with MONTH as a primary key (indicated with underline), because TIMEOFSHIP is separated from FSSFILE.

Since we delete SHIPMENTINFO (a pointer) from FSSFILE, we lost the relation with FSSSHIPFILE. We need to put a field to connect FSSFILE and FSSSHIPFILE. We can use the unique NUMBER in FSSFILE to be the foreign key in FSSSHIPFILE (indicated by postfixing a *).

The same thing happened with CONTAINER. Number can be composed from CONTAINER with InvoiceNo from FSSFILE as the primary key in CONTAINER to avoid those anomalies.

But here we have problems with 1NF. First, the redundancy, customer records are in FSSFILE, and one customer can have many sales information records in FSSFILE. So each time we have new sales contract with the same customer, we have to put in the redundant customer's data. Second, the insertion anomalies, new customers cannot be added until they have signed a sales contract with the company. Third, deletion anomalies, delete a sales contract could delete all the information of a customer if that customer has only one sales contract with the company.

So, we need to go through 2NF, to eliminate the dependencies mentioned above. We separate the customer record from FSSFILE. Now the relations are like the following:

CUSTOMER { Name, Addr, ContrNo }

FSSFILE { Number, ContrDate, Commodity, PriceBase,
Lc.Number, Lc.ExpDate, Lc.ShipDate, Lc.Bal, Lc.Amount,
TotalShip, BalofShip, IssueBank, DrawBank, MitiNo,
NofShipment }

FSSSHIPFILE { InvoiceNo, Name, Origin, Dest, Etd,
InvoiceDate, TotalBales, TotalNet, NofCont, Number* }

CONTAINER { Number, InvoiceNo, Bales, Net }

TIMEOFSHIP { Number, Month, Wgt, Bal, UnitPrice }

3. 3rd Normal Form

A relation is in 3NF if it is in 2NF and there is no transitive dependency. As we can see in the relations above, there is no transitive dependency. These relations are already in 3NF.

But, the CUSTOMER was just separated from FSSFILE, and there is no relation between CUSTOMER and FSSFILE. Perhaps the field CONTRNO in CUSTOMER can be used as a primary key. This could cause deletion anomalies, if we delete a record in FSSFILE, we might lose the record of CUSTOMER.

To avoid these problems, we can introduce the field NAME in CUSTOMER to be a foreign key in FSSFILE (indicated by postfixing with a *).

CUSTOMER { Name, Addr }

FSSFILE { Number, ContrDate, Commodity, PriceBase,
Lc.Number, Lc.ExpDate, Lc.ShipDate, Lc.Bal, Lc.Amount,
TotalShip, BalofShip, IssueBank, DrawBank, MitiNo,
NofShipment, Name* }

FSSSHIPFILE { InvoiceNo, Name, Origin, Dest, Etd,
InvoiceDate, TotalBales, TotalNet, NofCont, Number* }

CONTAINER { Number, InvoiceNo, Bales, Net }

TIMEOFSHIP { Number, Month, Wgt, Bal, UnitPrice }

We added one more field named PhoneNo to CUSTOMER to make it more useful. Then we renamed those file names and field names to make them more meaningful. The final result is:

Customer (cname, addr, phoneno)

S-Contract (snumber, contrdate, commodity, pricebase,
lcnumber, lcexpdate, lcshipdate, lcbal, lcamount
totalship, balofship, issuebank, drawbank, mitino,
nofshipment, cname*)

Timeofshipment (month, snumebr, wgt, bal, unitprice)

S_Shipment (invoiceno, name, origin, dest, etd, invoicedate,
totalbales, totalnet, nofcont, snumber*)

Container (cnumber, invoiceno, bales, net)

4. New E-R Diagram

Until now, we can draw a new Entity-Relational diagram (Figure 5-2) with the developed relations. As you can see the Entity-Relational diagram of the new program is much more simple and readable than that of the old program.

5. New Data Structures

Also, we used those relations to create data structures for the new program with the convention of dBASE III PLUS.

S-Contract

<u>Field Name</u>	<u>Type</u>	<u>Width</u>	<u>Dec</u>
-------------------	-------------	--------------	------------

ContrDate	Character	8	
Commodity	Character	50	
PriceBase	Character	80	
LcNumber	Character	12	
LcExpDate	Character	8	
LcShipDate	Character	8	
LcBal	Numeric	15	2
LcAmount	Numeric	15	2
TotalShip	Numeric	15	2
BalofShip	Numeric	15	2
IssueBank	Character	30	
DrawBank	Character	30	
MitiNo	Character	18	
NofShipment	Numeric	15	
CName	Character	25	

S-Shipment

<u>Field Name</u>	<u>Type</u>	<u>Width</u>	<u>Dec</u>
InvoiceNo	Character	15	
Name	Character	25	
Origin	Character	25	
Dest	Character	25	
Etd	Character	8	
InvoiceDate	Character	8	

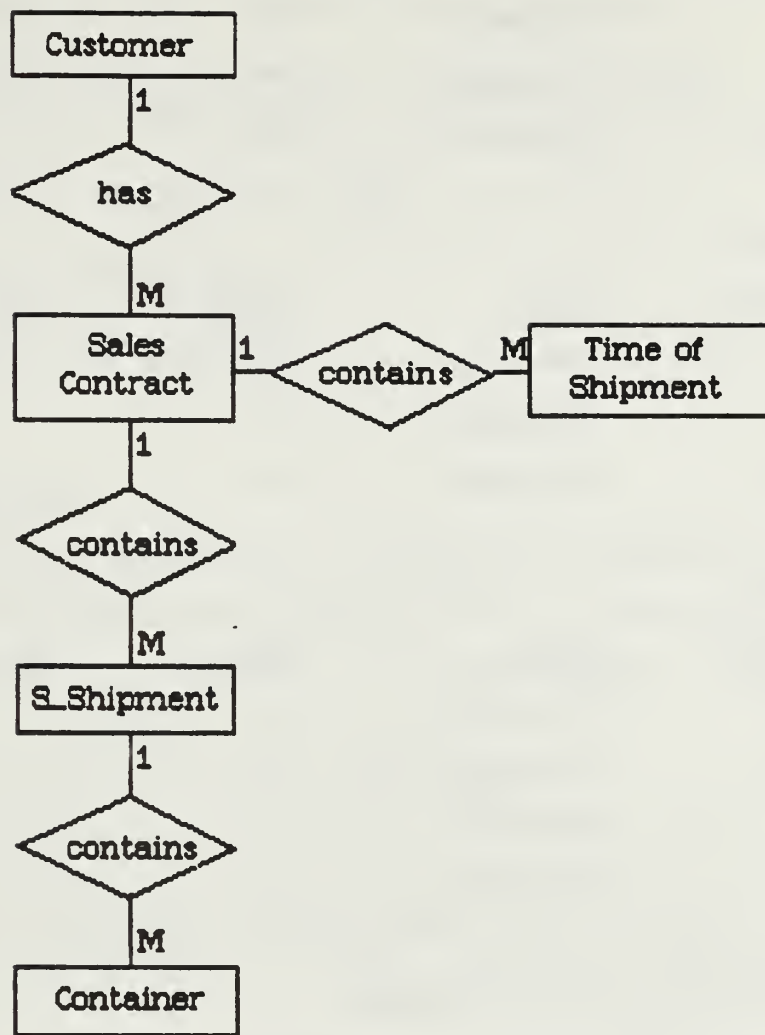


Figure 5-1 New E-R Diagram of Sales

TotalBales	Numeric	15	2
TotalNet	Numeric	15	2
NofContner	Numeric	15	
SNumber	Character	12	

TimeofShipment

<u>Field Name</u>	<u>Type</u>	<u>Width</u>	<u>Dec</u>
Month	Character	2	
SNumber	Character	12	

Wgt	Numeric	15	2
Bal	Numeric	15	2
UnitPrice	Numeric	15	2

Customer

<u>Field Name</u>	<u>Type</u>	<u>Width</u>	<u>Dec</u>
CName	Character	25	
Address	Character	50	
PhoneNo	Character	13	

Container

<u>Field Name</u>	<u>Type</u>	<u>Width</u>	<u>Dec</u>
CNumber	Character	12	
InvoiceNo	Character	15	
Bales	Numeric	15	2
Net	Numeric	15	2

B. NORMALIZATION OF PURCHASE

With all the procedures we applied to the **Sales**, we have a more simple, more structured relational database of **Sales**. We can applied these procedures to the **Purchase** too. The following are the results when we went through the **Purchase** with normalization, E-R diagram, and redefined data structures.

First, we study the file structures declared in the original program.

```
FSPFILE { Status, Number, ContrDate, Farmer.Name,
           Farmer.Addr, Commodity, TimeofShip, TotalShip, BalofShip,
           Nofshipment, ShipmentInfo }
```

```
TIMEOFSHIP { Month, Wgt, Bal, UnitPrice }
```

FSPHASHFILE { Status, Number, Name, Commodity, Link }

FSPSHIPFILE { Status, Truck, NofTruck, TotalBales, TotalNet,
Payment, TotalTruckCost, Link }

TRUCK { MthDay, WgtTicketNo, Bales, Net, Cost }

FSPCONTRACT { Same as FSPFILE }

FSPSHIPREC { Same as FSPSHIPFILE }

1. 1st Normal Form

For the same reasons in **Sales**, we deleted FSPCONTRACT and FSPSHIPREC, and eliminated fields STATUS, LINK, SHIPMENTINFO. Besides, the Number in FSPFILE is purchase number, so we substituted with the new field name PNumber.

Now, the new file structures look like these:

FSPFILE { PNumber, ContrDate, Farmer.Name, Farmer.Addr,
Commodity, TimeofShip, TotalShip, BalofShip, Nofshipment
}

TIMEOFSHIP { Month, Wgt, Bal, UnitPrice }

FSPSHIPFILE { Truck, NofTruck, TotalBales, TotalNet, Payment,
TotalTruckCost }

TRUCK { MthDay, WgtTicketNo, Bales, Net, Cost }

After the 1st Normal Form :

FSPFILE { PNumber, ContrDate, Farmer.Name, Farmer.Addr,
Commodity, TotalShip, BalofShip, Nofshipment }

TIMEOFSHIP { Month, Wgt, Bal, UnitPrice }

FSPSHIPFILE { NofTruck, TotalBales, TotalNet, Payment,
TotalTruckCost }

TRUCK { MthDay, WgtTicketNo, Bales, Net, Cost }

2. 2nd Normal Form

After the 2nd Normal Form and creating the new relations with newly separated files :

FARMER { Name, Addr }

FSPFILE { Number, ContrDate, Farmer.Name, Farmer.Addr,
Commodity, TotalShip, BalofShip, Nofshipment }

TIMEOFSHIP { PNumber, Month, Wgt, Bal, UnitPrice }

FSPSHIPFILE { PNumber, PShipNo, NofTruck, TotalBales, TotalNet,
Payment, TotalTruckCost }

TRUCK { PNumber, PShipNo, MthDay, WgtTicketNo, Bales, Net, Cost
}

3. 3rd Normal Form

Here we meet the same problem as we did in **Sale**. The FARMER just separated from FSPFILE, we created a new relation between FARMER and FSPFILE.

FARMER { Name, Addr }

FSPFILE { PNumber, ContrDate, Farmer.Name, Farmer.Addr,
Commodity, TotalShip, BalofShip, Nofshipment, Name* }

TIMEOFSHIP { PNumber, Month, Wgt, Bal, UnitPrice }

FSPSHIPFILE { PNumber, PShipNo, NofTruck, TotalBales, TotalNet,
Payment, TotalTruckCost }

TRUCK { PNumber, PShipNo, MthDay, WgtTicketNo, Bales, Net, Cost

}

Again, we put PhoneNo in FARMER to make it more flexible. We changed the file names and field names to make them more meaningful. Then, these are the final result:

Farmer (fname, addr, phoneno)

Purchase_Contract (pnumber, contrdate, commodity, totalship, balofship, nofshipment, fname*)

P_Shipment (pshipno, pnumber, noftruck, totalbales, totalnet, payment, totaltruckcost)

TimeOfShip (pnumber, month, wgt, bal, unitprice)

Truck (pshipno, pnumber, mthday, wgtticketno, bales, net, cost)

4. New E-R Diagram

With the new relations, we draw a new Entity-Relational diagram, which looks like the E-R diagram of **Sales**. (on next page)

5. New Data Structures

P-Contract

<u>Field Name</u>	<u>Type</u>	<u>Width</u>	<u>Dec</u>
PNumber	Character	12	
ContrDate	Character	8	
Commodity	Character	50	
TotalShip	Numeric	15	2
BalofShip	Numeric	15	2
NofShipment	Numeric	5	
FName	Character	25	

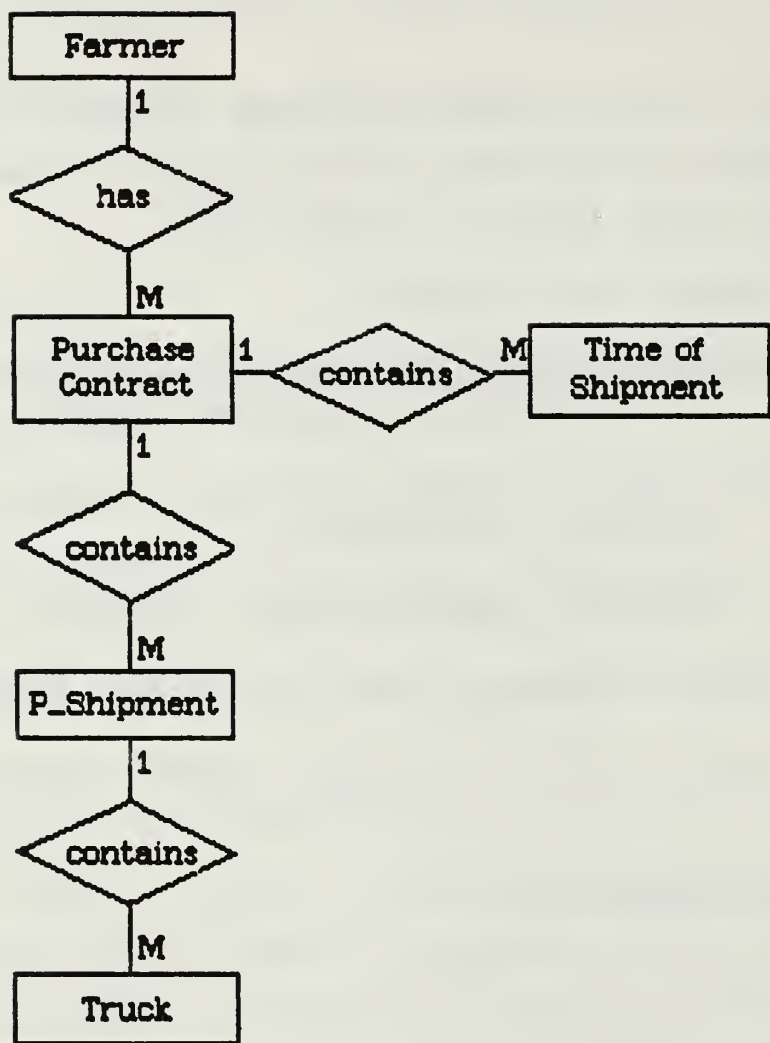


Figure 5-2 New E-R Diagram of Purchase

P-Shipment

<u>Field Name</u>	<u>Type</u>	<u>Width</u>	<u>Dec</u>
PShipNo	Character	12	
PNumber	Character	12	
NofTruck	Numeric	2	
TotalBales	Numeric	15	2
TotalNet	Numeric	15	2
Payment	Numeric	15	2

TruckCost	Numeric	15	2
-----------	---------	----	---

Farmer

<u>Field Name</u>	<u>Type</u>	<u>Width</u>	<u>Dec</u>
FName	Character	25	
Address	Character	50	
PhoneNo	Character	13	

Truck

<u>Field Name</u>	<u>Type</u>	<u>Width</u>	<u>Dec</u>
PShipNo	Character	12	
PNumber	Character	12	
MthDay	Character	8	
WgtTicketNo	Character	12	
Bales	Numeric	15	2
Net	Numeric	15	2
Cost	Numeric	15	2

VI. STUDY OF MAINTAINABILITY OF THESE TWO PROGRAMS

The term "maintainability" is used to describe the ability of a software activity that occurs following the delivery of that software product to the customer. A software product possesses the characteristic MAINTAINABILITY to the extent that it facilitates updating to satisfy new requirements.

A maintainable software product is one which is understandable, testable, and easy to modify. One must be able to modify the product to rectify a deficiency or to add new capabilities or to allow a program to operate on a different computer system.

The maintenance phase of the software life cycle is the time period in which a software product performs useful work. Typically, the maintenance phase in the life cycle for a software product spans longer than that for the development phase.

Maintainability concerning the ability of a software being making enhancements to that software products, adapting products to new environments, and correcting problems. Software product enhancements can involve providing new functional capabilities, improving user displays, and modes of interaction, or upgrading the performance characteristics of a system. Adaption of software to a new environment can involve moving the software to a different machine. Problem correction involves modification and revalidation of software to correct errors. Also, the changing requirements from users can cost much in maintaining software.

In most software engineering studies, it is shown that maintainence activities actually break into three main subactivities. They are : **Perfective Maintenance, Adaptive Maintenance, and Corrective Maintenance.**

Perfective maintenance is the act of improving the software's function by responding to customer- and programmer-defined changes. This is not the portion of software maintenance that is involved with fixing errors. But it is the biggest maintenance time consumer.

Adaptive maintenance is the act of changing software to adapt to environmental changes. If the computer on which the software runs is going to get a new version of the operating system, or the total system data base must have some detail level changes (for example, if the ZIP code is increased from five to nine digits), the software must be adapted to meet those changes.

Corrective maintenance is the act of keeping software free from errors and guarantees the integrity of data.

A. THE ORIGINAL PROGRAM

1. Maintainability of Record Structures

In the original program, the data structures used in defining files are records. As we know that Pascal has very strong typing, this means that the records used to define files have domain constraints. A domain constraint simply states that values of the attribute in question are required to belong to the set of values constituting the underlying domain. Here we discuss the maintainability of the original program and the new program by checking the record structures and control structures.

a. Add Fields

Referring to the records and file declarations of the original program in Appendix A, we can summarize them as follow:

SALE'S FILE DECLARATIONS

```

fssfile : FILE OF fsstype;
fsshashfile : FILE OF entry;
fssshipfile : FILE OF shiptype;
fsscontract, fssdummy : fsstype;
fssshiprec : shiptype;

```

Now, there are one file (i.e., fssfile) and two file buffers (i.e., fsscontract, fssdummy) using the same data structures:

```

fsstype = PACKER RECORD
    status : statustype;
    number : string[12];
    contrdate : datatype;
    customer : customertype;
    commodity : string[50];
    pricebase : string[80];
    lc      : lctype;
    timeofship : ARRAY[1..7] OF tostype;
    totalship, balofship : real;
    issuebank, drawbank : string[30];
    mitino : string[18];
    nofshipment : integer;
    shipmentinfo : integer;
END;

```

Within this record structure, there are five different user-defined data types:

```

statustype = (occupied, empty);

```

```

datatype = PACKED RECORD
    month,
    day,
    year : integer;
END;

```

```

customtype = PACKED RECORD
    name : string[25];
    addr : string[50];
    contrno : string[12];
END;

```

```

lctype  = PACKED RECORD
        number : string[12];
        expdate,
        shipdate : datatype;
        bal, amount : real;
END;

```

```

tostype = PACKED RECORD
        month, wgt : integer;
        bal, unitprice : real;
END;

```

What is interesting is that inside **lctype**, there is a **datatype**. Now **datatype** has been referenced in a different hierarchical level. Suppose we are going to put a new field, say **TIME**, in **datatype**, and suppose that we only use **TIME** in **contrdate**, not in **lctype**. This causes some troubles. First, we have to define a new type, say **datetimetype**, which has an extra field that contains the time of the event. Second, we need to write another user-defined function to check **datetimetype** besides function **DATECHECK**. Third, we need one more procedure to convert **datetimetype** besides procedure **DATECONVERT**. Fourth, after all this has been done, we need to recompile the source program, redebug, recompile, redebug, and so on. Also, within **FSSNEW**, in **READNEXTINPUT**, we have to modify statements to meet the new data types.

The same problems occur if we want to add one field, say phone number, to **customtype**.

Now, let's have a look at another file structure in **Sale**, which is referenced by **fssshipfile** and **fssshiprec** :

```

shiptype = PACKED RECORD
        status : statustype;
        involveno : string[15];
        name,
        origin,
        dest : string[25];

```

```

        etd,
        invoicedate : datatype;
        totalbales : integer;
        totalnet : real;
        container : ARRAY[1..20] OF contype;
        nofcont : 0..20;
        link : integer;
    END;

```

In this file type, three user-defined data types being referenced :

```

statustype = (occupied, empty);

```

```

datatype = PACKED RECORD
    month,
    day,
    year : integer;
END;

```

```

contype = PACKED RECORD
    number : string[12];
    bales : integer;
    net : real;
END;

```

The same situations have to be considered if we want to add fields into these three data types or the file type.

b. Delete Fields

What will happen if we delete the field **year** from **datatype**, or delete **contrno** from **customtype** ? How much trouble can it cause ? The same as above ? Will we lose all the data kept in that field? Of course. There is no way to prevent this, even an experienced programmer uses structured-programming techniques such as Information Hiding. The reason is, Pascal was designed for general programming environments, peoples can use Pascal to design an Operating System, or a compiler, or a database system or any other programs that meet scientific or business requirements.

Same problems happen with **Purchase**. Let's look at Appendix A :

PURCHASE'S FILE DECLARATIONS

```
fspfile : FILE OF fsptype;  
fsphashfile : FILE OF entry;  
fspshipfile : FILE OF fspshiptype;  
fspcontract : fsptype;  
fspshiprec : fspshiptype;
```

Again, there are one file (i.e., fspfile) and one file buffer (i.e., fspcontract) using the same file type :

```
fsptype = PACKED RECORD  
    status : statustype;  
    number : string[12];  
    contrdate : datatype;  
    farmer : farmertype;  
    commodity : string[50];  
    timeofship : ARRAY[1..7] OF tostype;  
    totalship,  
    balofship : real;  
    nofshipment,  
    shipmentinfo : integer;  
END;
```

Within this record type, there are five different user-defined data types :

```
statustype = (occupied, empty);
```

```
datatype = PACKED RECORD  
    month,  
    day,  
    year : integer;  
END;
```

```
farmertype = PACKED RECORD  
    name : string[25];  
    addr : string[50];
```

END;

```
totype = PACKED RECORD
        month,wgt : integer;
        bal,unitprice : real;
END;
```

The same discussions can be used here as we used in **Sales**. We have the same conclusions.

B. THE NEW PROGRAM

dBASE III PLUS is a versatile database manager. It is designed to handle the many business applications in which the user needs to manage large amounts of repetitive information. It can function as a simple file manager, it can handle the complex issues in relating files to one another, and it can be used as a complete computer language like Pascal or BASIC. Its fast development means that appropriate jobs can be completed in much less time with dBASE III PLUS than with Pascal or BASIC. It is possible to stop and start dBASE III PLUS between instructions and so have a microscope into the commands. Pascal and BASIC are compiled, so if things go wrong, it is very hard to discover where the mistake was made. In dBASE III PLUS, when changes are made the results appear instantaneously. From the dBASE III PLUS prompt, it is possible to go anywhere and see anything. Everything is open to instant inspection and change. This kind of power can be its own problem. dBASE III PLUS has the disadvantage of being a highly accessible language. You can always single step your way through your command files, make never ending, instantaneous changes with MODIFY COMMAND, etc.

dBASE III PLUS has sophisticated yet easy indexing. This means

that a list can be organized in several different ways simultaneously and that any item can be found in a fraction of a second in a small list (says less than 2000 records) and only a few seconds in a list of many thousands of records.

dBASE III PLUS is far from a perfect programming system. Frequently there are bugs where the program simply does not do what it is supposed to do.

1. Maintainability of Record Structures

a. Add Fields

dBASE III PLUS fully supports the **Relational Data Model**. The files are created as two dimensional tables. At dBASE III PLUS dot prompt, just type in

```
CREATE <filename>
```

The effect of this action is that dBASE III PLUS sets up a screen layout, expecting your file-definition entries. You just provide information for that file. At this point, dBASE III PLUS wants you to specify the **name** of each field you want to define, the **type** of the field, the **length** of the field, and the number of **decimal** places, if the field is a numeric field. Here is an example of file structure CUSTOMER used in **sale** :

```
Structure for database : A:customer.dbf
Number of data records:      0
Date of last update       : 05/01/87
Field   Field Name      Type      Width   Dec
1       CNAME           Character 25
2       ADDRESS          Character 50
3       PHONENO          Character 13
** Total **                89
```

Perhaps you want to change the structure of one of the existing fields, (maybe a name change, a type change, or a length change); or perhaps you want to add one more field into the

structure. Regardless of the kind of change you want, it would be logical to make changes anytime. This does not mean, however, that you cannot alter the structure of an existing database that contains data records. You can change structures at any time, in any database.

To add a field in this file structure, enter :

.USE CUSTOMER <cr> (Not necessary if the file is
already in use)

.MODIFY STRUCTURE <cr> or

.MODI STRU

This brings up the structure of the file on the screen. Then use the combination of cursor controls to add a new field. When you have made the required changes to your structure, you can either enter Ctrl-W or Ctrl-END to save the new structure, or enter Ctrl-Q or ESC to change your mind on the changes made. You don't have to re-compile the program.

Of course, if you have just created a new file and there are no data records in it, you can modify its structure at will, without any adverse effects. Yet, the most important consideration - Future change. Future changes to data files in a database should be anticipated as much as possible in order to buffer the amount of restructuring required. Future changes encompass three areas : hidden keys, addition of dependent fields, and high-usage of a secondary key.

Hidden Keys

Each of the dependent fields in a data file should be examined to determine whether any might become key fields in the future. If they are identified, they can either be removed from the data file and added to a new data file, or left in the

existing data file, as long as flexibility is provided to accommodate future key field changes.

Addition of Dependent Fields

The data file structures should have room for the addition of dependent fields. Even though this will require reloading of the records whenever new fields are added, it should not affect the application program written.

High-Usage of Secondary Key

If you anticipate that a dependent field acting on occasion as a key field will be used more in the future, the data file structures and application programs should be designed to accomodate change. This may require the dividing of the data file into two data files at the very start, or the writing of the application programs to accept this anticipated future change.

b. Delete Fields

To delete a field from data file in database, the above strategies can be applied. But when we delete a field, the data retained for that field will be gone, unless you rename the field name or copy that record data to some temporary store areas.

To delete a field, we have to be cautious about if it is a primary key of a file or not. dBASE III PLUS has a very important feature called **INDEXING**. Indexing is an inherent part of the dBASE III PLUS scenario. If your goal is to write sophisticated application programs, you cannot do so without the indexing feature!

In the process of INDEXing, you inform dBASE III PLUS of

your intention to create an **index file** on one or more of the fields of the master file you are working with.

Index file is created as follow :

```
.USE CUSTOMER      <cr>
```

```
.INDEX ON CNAME TO CUSTINDX <cr>
```

This results in the creation of a separate file called an **index file**, whose name is CUSTINDX. You can provide any primary name you want. dBASE III PLUS provides the default extension of .NDX. An index file is just an index file. It is not a dBASE III PLUS database. It only contains pointers to the actual records in the data files.

Now, suppose you want to delete the field called CNAME! What is going to happen? How can CUSTINDX index on? Will you lose all the data records? Without that field which the index file indexed on, you cannot manipulate that file, like DISPLAY, LIST, PRINT, etc,. You have lost them. So, before you delete CNAME you might want to re-index the CUSTOMER on another field. You can then delete CNAME.

VII. STUDY OF MODIFIABILITY OF THESE TWO PROGRAMS

A. MODIFIABILITY OF PASCAL

1. Modifying Data Fields in Pascal Program

Pascal includes the ability to name **constants** of discrete types (integers, enumerations, and characters) which aid the modifiability. Constant declarations are introduced by the word **const** and have the syntax

`<name> = <constant>;`

Why do the constant declarations help modifiability?

Consider a large program that manipulates arrays. In such a program there will normally be many dependencies between the dimensions of these arrays and other parts of the program. For example, if the arrays have dimensions [1..100] then there will normally be **for**-loops with bounds "1 to 100" or "100 **down to** 1". Further, there may be other arrays that have dimensions [1..100], [0..99], and so forth, in order to be compatible with the first array. All of these interdependencies are implicit, although proper documentation can help to make them explicit.

Next, suppose that in the normal process of modifying this program, it is decided that the size of one of these arrays should be changed to [1..150] (say to accomodate larger data sets). This will imply that the dimensions of some of the other arrays also be changed and that the limits on some of the **for**-loops be changed. The programmer will have to find all numbers that implicitly depend on the changed dimension and make appropriate changes to these numbers. It is not as simple as using an editor to change all "100"s to "150"s since some "100"s depend on the array's dimensions and some don't. Rather, each instance of "100" must be considered

individually. This is a very error-prone and tedious process.

Pascal's constant declarations solve many of these problems since they allow implicit dependencies to be made explicit. They do this through the application of the Abstraction Principle since all the dependent constants are abstracted out of the program and are given a name. For example, given the declaration

```
const MaxData = 100;
```

The programmer can use the name constant "MaxData" in all declarations, **for**-loop, and so on, that depend on this parameter. Changing this parameter to 150 is then simple since only the constant declaration has to be changed; all other dependent changes are made automatically. Here in the original program two similar constant are used :

```
max = 30;
```

```
pmax = 30;
```

These constants are used to define the size of hash tables, one in **Sale**; one in **Purchase**. These might aid the modifiability of original program. yet, there are some other declarations which do not take advantage of **constant** declaration. For example, the **timeofship** has been defined as an array of [1..7] of **tostype**, the same to **truck** and **container** they are all defined as an array of [1..20]. Suppose, one day the number of truck or container is going to expand to more than 20, we have to go to declarations and change the values and all those implicit dependencies.

Nevertheless, if we expanded the size of an array or a table, or if we shrink them, is there any mechanism in Pascal which can help us to back up the data stored in old arrays?

If there is no such mechanism, then the programmers have to take care of all this mess. Maybe need to write another program to do this job. Then compile and debug and so on.

2. Modifying Functions in Pascal Program

Pascal is a block-structured language, and a block-structured language requires the development of new run-time techniques. Since Pascal procedures can be recursive, there can be several instances of a procedure active at one time. Hence there must be some provision for the dynamic creation of activation records to hold the state of these instances. Therefore, the static "one activation record per procedure" techniques will not work. Also, we have seen that Pascal provides dynamic memory management by allocating space for the locals of a procedure on a stack. This storage is allocated on procedure entry and deallocated on procedure exit. This means that variables cannot be statically bound to memory locations as is common in FORTRAN and assembly languages. In Pascal, activation records represent the state of an activation. Procedures require both static and dynamic links. The static link is set to the environment of definition and the dynamic link points to its context. Procedural parameters are represented by closures. Pascal, Algol, and many other languages allow procedures and functions to be passed as arguments to other procedures and functions. Allowing functions to accept and return other functions leads to a very powerful style of programming, called **functional programming**. Functional programming languages must use a different discipline for the allocation and deallocation of activation records.

So far, we have studied the various of data typing, various discipline of memory allocation and various considerations of procedure calling and return. We know once we have developed a program for one application, "its tied up to that application. It is very difficult to add one function or eliminate one function from

that program. Because, all the procedures are dependent on each other, and all the data declarations, all the programming codes was so program dependent.

B. MODIFIABILITY OF dBASE III PLUS

1. Modifying Data Fields in dBASE III PLUS Program

If you are modifying the structure of an existing database that has data-records in it, note that you should choose your modification cautiously. Consider the following situation. Suppose we have a character field called SNUMBER that has, as data, a combination of digits and characters, starting with digits, and you want to change its type to numeric. Since dBASE III PLUS will not retain character data in a numeric field, at the end of the modification, we will have lost all the character data from that SNUMBER field! Only the leading numerics will be retained in the (new) numeric field. If the original data had leading characters instead of numerics, nothing would have been retained.

If you change a field name and a field length at the same time, (either in the same or different fields), note that we will lose the data for the field(s) with the name change! Since dBASE III PLUS cannot handle this dual-change at the same time, make one of the changes first (either one) and then make the other change to modify the structure.

If you change the name of a field (only), and you want to save this new structure, dBASE III PLUS will ask you a question : "Should data be COPIED from backup for all fields?(Y/N)". What this means is that, by default, when you change the name of a field, you will lose all your data from all the records for the specific field, at the end of modification. By responding with a Y to this question, you can retain all your data for all the records for that

field, at the end of name-change modification.

To modify the structure of the database you have created, just enter at the dBASE III PLUS dot prompt :

```
.USE CUSTOMER      <cr>
.MODIFY STRUCTURE   <cr>      or
.MODI STRU          <cr>
```

MODIFY COMMAND is dBASE III PLUS' text editor. It is crude by text-editing standards, but it has one major advantage: it can be accessed directly from dBASE III PLUS.

Part of dBASE III PLUS' appeal is that changes can be made to a program very quickly. The MODIFY COMMAND allows programmers to alter command files and test the changes in as fast a way as possible. It is especially useful after a program is written, when the program then needs to be debugged. During this stage simple bugs, spelling mistakes, and syntax errors predominate. These usually require only a little thought, and the ability to get to a text editor quickly is extremely valuable.

Because it is a very simple text editor, the more a change needs sophisticated editing, the less appealing MODIFY COMMAND will be. If the programmer wishes to move commands from one place to another, to copy groups of command, or to search the command file, then MODIFY COMMAND is a poor tool.

The MODIFY COMMAND has about 20 commands. It uses WordStar-like commands. There are two major problems with MODIFY COMMAND (in addition to all the features it lacks). First, it only allows command files of about 4,096 characters. If programmers go beyond this limit, then a message will be received saying that data will be lost if they try to save the file. Second, if the user deletes a line or if a character is inserted into a large file, it takes an annoyingly long time to rewrite the screen.

By today's standards, the MODIFY COMMAND is crude, and many find it a source of irritation. But as an old dBASE and CBASIC programmer, the MODIFY COMMAND was revolutionary when it first appeared. Prior to that, all programs had to exit from their program (e.g., Pascal), go to the Operating System, then to their word processor, again to the Operating System, and then back to the program. Each change in a program required the same lengthy journey. All this was obviated by the MODIFY COMMAND.

2. Modifying Functions in dBASE III PLUS Program

Here what we discuss is how to add or delete an application to or from a database. There are tremendous benefits to be gained by subdividing a problem into several command files. But there are some differences between writing a single command file and writing a project in several command files. These differences only concern memory variables in dBASE III PLUS; they involve no restrictions on data and index files.

The major benefit is that thinking can be organized by breaking the job into a series of tasks, each of which is performed by a command file. This makes development and testing easier, and makes adding and deleting functions or applications easier. Provided that the task of dividing the main task into subtasks has been properly thought through, the program will be more flexible and changes will be easier to implement than if the entire task was performed in one command file.

There is a convention that a command file that calls another command file is at a higher level. The command file that is called is at a lower level than the command file calling it. dBASE III PLUS can move up or down the structure of command files, to higher or lower level command files. It cannot move sideways.

When a task is performed across a number of command files, dBASE III PLUS generally will perform more slowly than if all commands were in one file. This is because each command file has to be opened by DOS when a DO (a command in dBASE III PLUS) is performed and closed by DOS when a RETURN is performed. The time taken for this can be considerable. SET PROCEDURE will circumvent the opening and closing of files by DOS.

In the new program, we subdivided the old program into several subprograms and tied up with the E-R diagram of the new program in five levels (see Figure 5-2 and Figure 5-3), they are:

HANAOKA

SALE

NEW SALE CONTRACT

SALE INFORMATION INQUERY

NEW SALE SHIPMENT

SCOMPUTE

SALE INFORMATION INQUERY

LIST CUSTOMER

LIST ALL CONTRACTS OF ONE CUSTOMER

LIST ONE CONTRACT INFORMATION

SALE LIST

PURCHASE

NEW PURCHASE CONTRACT

PURCHASE INFORMATION INQUERY

NEW PURCHASE SHIPMENT

PCOMPUTE

PURCHASE INFORMATION INQUERY

PURCHASE LIST

PURCHASE SHIPMENT LIST

The name at each line represents a command file. Totally

there are 19 command files in the new program. All the command files are independent of each other. One command file has one function only, so it's easy for programmer to delete a command file if that function or application is no longer necessary. The programmer can add a function or application to this program by creating a command file for that function or application.

VIII. CONCLUSIONS

The increasing productivity of systems development, the shortening of the response time of the computer, and the increasing complexity of computer systems, requires effective tools that will be capable of processing information. Many changes have occurred in the last five years. These changes are of two fundamental types. First, people have learned how to better manage and use database technology. Five years ago, companies were still wrestling with databases. With no powerful database language available, they used generic programming languages to implement databases. To use generic programming languages to simulate relational database model or for any Database Management System is quite a heavy job. Some advantages of a database must be sacrificed due to the lack of database features.

In 1981, this situation changed. Some of the database languages, based on the relational model, were announced as products. Early relational products had unacceptable performance. In the last two years, performance has been improved. Major manufacturing organizations have tested relational DBMS products and have found segments of their workload that can be processed with acceptable performance. Using a relational DBMS, one company found application development productivity improvements to be greater than 100 to 1.

In this study, we examined two programs, one was written in Pascal, and the other was written in dBASE III PLUS. We can compare the lines of the source code of these two programs (see Appendix A and B). The original program is quite a big program, and due to the programming structure it is difficult to read. In the

new program, we see that its clear and it is well-sequenced so it is easier to read.

In comparing their maintainability and modifiability, the new program is much easy to maintain, and easy to modify.

APPENDIX A
THE ORIGINAL PROGRAM (WRITTEN IN APPLE PASCAL)

```
(*$$+*)
PROGRAM hanaoka;
CONST ok = true;
      x  = 10;
      max = 30; (* size of hash table *)
      pmax = 30;

TYPE characters = string[80];
   numbers  = PACKED ARRAY[1..13] OF integer;
   menutype = PACKED ARRAY[0..9] OF characters;
   intype   = PACKED ARRAY[0..20] OF characters;
   datatype = PACKED RECORD
               month,
               day   ,
               year  :integer;
           END;

   statustype = (occupied, empty);

   tostype    = PACKED RECORD
               month, wgt   : integer;
               bal, unitprice : real
           end;

   contype    = PACKED RECORD
               number : string[12];
               bales   : integer;
               net : real {integer but larger than maxint}
           END;

   customtype = PACKED RECORD
               name : string[25];
               addr : string[50];
               contrno : string[12]; (* their contract no *)
           END;

   lctype     = PACKED RECORD
               number : string[12];
               expdate,
               shipdate : datatype;
               bal, amount : real
```

END;

fsstype = PACKED RECORD

status : statustype;
number : string[12];
contrdate : datatype;
customer : customtype;
commodity : string[50];
pricebase : string[80];
lc : lctype;
timeofship : ARRAY[1..7] of tostype;
totalship, balofship : real;
issuebank, drawbank : string[30];
mitino : string[18];
nofshipment,
shipmentinfo: integer;
(* ptr to the shipments info *)

END;

shiptype = PACKED RECORD

status : statustype;
invoiceno : string[15];
name, origin, dest : string[25];
etd, invoicedate : datatype;
totalbales : integer;
totalnet : real;
container : ARRAY[1..20] OF contype;
nofcont : 0..20;
link : integer;

END;

entry = PACKED RECORD

status : statustype;
number : string[12];
name : string[25];
commodity : string[50];
link : integer

END;

farmertype = PACKED RECORD

name : string[25];
addr : string[50]

END;

trucktype = PACKED RECORD

```

        mthday : string[6];
        wgtticketno : string[12];
        bales : integer;
        net,cost : real;
    END;

```

```

fspstype = PACKED RECORD

```

```

    status : statustype;
    number : string[12];
    contrdate : datatype;
    farmer : farmertype;
    commodity : string[50];
    timeofship : ARRAY[1..7] OF tostype;
    totalship,balofship : real;
    nofshipment,
    shipmentinfo : integer;

```

```

    END;

```

```

fspshiptype = PACKED RECORD

```

```

    status : statustype;
    truck : ARRAY[1..20] OF trucktype;
    noftruck : 0..20;
    totalbales : integer;
    totalnet,
    payment,
    totaltruckcost : real;
    link : integer

```

```

    END;

```

```

VAR  fspfile      : FILE OF fspstype;
     fsphashfile  : FILE OF entry;
     fspshipfile  : FILE OF fspshiptype;
     fspcontract  : fspstype;
     fspshiprecord : fspshiptype;
     fssquerymenu, error1, error2, fssshpmenu,
     sciname, fdstfmenu, fssmenu, fspmenu,
     fspquerymenu, fspshpmenu, fspnewconmenu : menutype;
     fssnewconmenu : intype;
     choice : integer;
     menu : TEXT;
     out : INTERACTIVE;
     digit : SET OF '0'..'9';
     continue, quit : boolean;
     fssfile : FILE OF fsstype;
     fsshashfile : FILE OF entry;
     fssshipfile : FILE OF shiptype;

```

```

fsscontract, fssdummy : fsstype;
contractno : characters;
shpamtintons, rate : real;
fssshiprec : shiptype;

```

```

FUNCTION at(i,j:integer):char:FORWARD;
FUNCTION password:boolean;FORWARD;
FUNCTION datecheck(date:datatype):boolean;FORWARD;
FUNCTION conint(line:characters):integer;FORWARD;
FUNCTION conreal(line:characters):real;FORWARD;
FUNCTION valldate(name:characters):boolean;FORWARD;
PROCEDURE addcomma(VAR line:characters);FORWARD;
PROCEDURE dollarcent(num:real;VAR twodeci:characters);
    FORWARD;
PROCEDURE prealtostr(num:real;VAR twodeci:characters);
    FORWARD;
PROCEDURE skip(n:integer);FORWARD;
PROCEDURE dateconvert(line:characters;VAR date:datatype);
    FORWARD;
PROCEDURE prompt(list:menutype;n:integer; VAR select:integer);
    FORWARD;

```

```

(*$I #5:fss1.text *)
(*$I #5:fss2.text *)
(*$I #5:fss2.5.text *)
(*$I #5:fss3.text *)
(*$I #5:fsp1.text *)
(*$I #5:fsp2.text *)

```

```

FUNCTION at;
BEGIN
    GOTOXY(i,j);
    at := chr(0)
END;

```

```

FUNCTION validate;
VAR ch:char;i,max:integer;okset:SET OF char;
BEGIN
    max := LENGTH(name);
    okset := ['A'..'Z'] + ['a'..'z'] + [' ',' ',' ',' '];
    validate := false;
    FOR i := 1 TO max DO
    BEGIN
        ch := name[i];
        IF NOT(ch IN okset) THEN EXIT (validate)
    END;

```

```

END;
validate := true;
END;

```

```

FUNCTION conint;
VAR i,max,temp:integer; item:char;
BEGIN
    max := LENGTH(line); i := 0; temp := 0;
    REPEAT
        i := i + 1;
        item := line[i];
        IF item IN digit
            THEN temp := 10*temp+ord(item)-ord('0')
    UNTIL (item = '.') OR (i = max);
    conint := temp
END;

```

```

FUNCTION conreal;
VAR i,j,max : integer; temp : real; item : char;
BEGIN
    i := 1; temp := 0; max := LENGTH(line);
    line := CONCAT(line, ' '); {append one blank so not to give the
                                value range error for no. with no dec pt}
    WHILE (i <= max) AND (line[i] <> '.') DO
        BEGIN
            item := line[i];
            IF item IN digit
                THEN temp := 10*temp+ord(item)-ord('0');
            i := i + 1
        END;
    j := 10; i := i + 1;
    (* convert the decimal places if any *)
    WHILE (i <= max) DO
        BEGIN
            item := line[i];
            IF item IN digit
                THEN temp := temp + (ord(item) - ord('0'))/j;
            j := j * 10;
            i := i + 1;
        END;
    conreal := temp;
END; (* conreal *)

```

```

PROCEDURE skip;
VAR i : integer;

```

```

BEGIN
  FOR i := 1 TO n DO
    writeln(out)
  END;

PROCEDURE prealtostr;
{to change pseudo-real (i.e. integer>maxint) to string}
VAR number : ARRAY[0..9] OF string[1];
    tento   : ARRAY[0..5] OF real;
    i,j      : integer;
    temp     : real;
BEGIN
  number[0] := '0'; number[1] := '1'; number[2] := '2';
  number[3] := '3'; number[4] := '4'; number[5] := '5';
  number[6] := '6'; number[7] := '7'; number[8] := '8';
  number[9] := '9';
  tento[0] := 1; tento[1] := 10; tento[2] := 100; tento[3] := 1000;
  tento[4] := 10000; tento[5] := 100000.0;
  twodeci := ' ';

  FOR j := 5 DOWNTO 0 DO
    BEGIN
      temp := TRUNC(temp);
      twodeci := CONCAT(twodeci, number[i]);
      num := num - tento[j]*i
    END;

    {delete the leading zeroes}
    WHILE twodeci[i] = '0' DO
      DELETE(twodeci, 1, 1);
    END; {prealtostr}

PROCEDURE dollarcent;
VAR whole, dec: string[6];
    number : ARRAY[0..9] OF string[1];
    tento   : ARRAY[0..5] OF real;
    i,j      : integer;
    temp     : real;
    negative: boolean;
BEGIN
  number[0] := '0'; number[1] := '1'; number[2] := '2';
  number[3] := '3'; number[4] := '4'; number[5] := '5';
  number[6] := '6'; number[7] := '7'; number[8] := '8';
  number[9] := '9';
  tento[0] := 1; tento[1] := 10; tento[2] := 100; tento[3] := 1000;

```

```
tento[4] := 10000; tento[5] := 100000.0;
whole := ' '; dec := ' ';
```

```
IF num < 0
  THEN BEGIN
    negative := true;
    num := -1 * num;
  END
  ELSE negative := false;
FOR j := 5 DOWNTO 0 DO
  BEGIN
    temp := num / tento[j];
    i := TRUNC(temp);
    whole := CONCAT(whole, number[i]);
    num := num - tento[j]*i
  END;
{delete the leading zeroes}
WHILE whole[1] = '0' DO
  DELETE(whole, 1, 1);
num := TRUNC(num * 100.0 + 0.5);
FOR j := 1 DOWNTO 0 DO
  BEGIN
    temp := num/tento[j];
    i := TRUNC(temp);
    dec := CONCAT(dec, number[i]);
    num := num - tento[j]*i;
  END;
IF negative
  THEN twodeci := CONCAT('-', whole, '.', dec)
  ELSE twodeci := CONCAT(whole, '.', dec)
END;
```

```
PROCEDURE addcomma;
VAR i : integer;
BEGIN
  IF POS('.', line) = 0
    THEN i := 2
    ELSE i := 5;
  WHILE i < LENGTH(line) - 1 DO
    BEGIN
      INSERT(',', line, LENGTH(line) - i);
      i := i + 4;
    END
  END;
END;
```

```

PROCEDURE dateconvert;
VAR temp:datatype; max,i:integer; item :char;
BEGIN
    max := LENGTH(line);
    temp.month := 0;temp.day := 0;temp.year := 0;
    i := 1;
    WHILE line[i] <> '/' DO
        BEGIN
            item := line[i];
            temp.month := temp.month * 10 + ord(item) - ord('0');
            i := i + 1;
        END;
        i := i + 1;
        WHILE line[i] <> '/' DO
            BEGIN
                item.day := temp.day * 10 + ord(item) - ord('0');
                i := i + 1;
            END;
            i := i + 1;
            WHILE i <= max DO
                BEGIN
                    item := line[i];
                    temp.year := temp.year * 10 + ord(item) - ord('0');
                    i := i + 1;
                END;
                date := temp
            END; (* dateconvert *)

```

```

FUNCTION datecheck;
BEGIN
    IF (date.month < 1) OR (date.month > 12)
        THEN datecheck := false
    ELSE IF (date.day < 1) OR (date.day > 31)
        THEN datecheck := false
    ELSE IF (date.year < 60) OR (date.year > 99)
        THEN datecheck := false
    ELSE    datecheck := true
END; (* datecheck *)

```

```

PROCEDURE prompt;
VAR ch:char; i:integer;
BEGIN
    write(chr(12),at(15,1),list[0]);
    FOR i := 1 TO n DO
        write(at(10,2*(i+1),i,'. ',list[i]));

```

```

write(at(10,2*(n+2)), 'Selection Please:');
read(KEYBOARD, ch);

WHILE (ord(ch) < ord('1')) OR (ord(ch) > n + ord('0')) DO
BEGIN  (* range check for the selection input *)
    write(at(10,2*(n+2)), 'Invalid selection', chr(7), chr(7));
    write(at(10,2*(n+3)), 'Selection Please:');
    read(KEYBOARD, ch)
END; (* WHILE *)
select := ord(ch) - ord('0')  (* convert to integer *)
END; (* prompt *)

FUNCTION password;
CONST valid = 'hanaoka';
VAR cnt: integer; secret : string[7];
BEGIN
    cnt := 0;
    REPEAT
        write(chr(12), chr(7), at(10,5), 'Password :');
        readln(KEYBOARD, secret);
        cnt := cnt + 1;
    UNTIL (secret = valid) OR (cnt = 3);
    IF secret <> valid
        THEN password := false
        ELSE password := true
    END; (* password *)

PROCEDURE fdstfpurchase;
VAR quit, successful : boolean;
    choice : integer;
BEGIN
    (*$I-*)
    REPEAT
        RESET(fsphashfile, '#5:fsphashfile');
        successful := (IORESULT=0);
        IF NOT successful
            THEN BEGIN
                write(chr(12), chr(7), at(0,4), 'Wrong data diskette in
                    drive 2');
                write(at(5,0), 'put correct one and press <RETURN>');
                readln
            END
        ELSE CLOSE(fsphashfile)
    UNTIL successful;
    (*$I+*)

```

```

quit := false;
REPEAT
    prompt(fspmenu, 4, choice);
    CASE choice OF
        1 : fspnew;
        2 : fspshipment;
        3 : fspinquery;
        4 : quit := true
    END;
UNTIL quit
END; (* fdstfpurchase *)

PROCEDURE fdstfsales;
VAR quit, successful : boolean;
    choice : integer;
BEGIN
    (*$I-*)
    REPEAT
        RESET(fsshashfile, '#5:fsshashfile');
        successful := (IORESULT=0);
        IF NOT successful
            THEN BEGIN
                write(chr(12), chr(7), at(0,4), 'Wrong data diskette in
                    Drive 2');
                write(at(0,5), 'put correct one and press <RETURN>');
                readln
            END
        ELSE CLOSE(fsshashfile)
    UNTIL successful;
    (*$I+*)
    quit := false;
    REPEAT
        prompt(fssmenu, 4, choice);
        CASE choice OF
            1 : fssnew;
            2 : fssshipment;
            3 : fssinquery;
            4 : quit := true
        END;
    UNTIL quit
END; {fdstfsales}

PROCEDURE initialize;
VAR i:integer;
BEGIN

```

```

quit := false;
digit := ['0','1','2','3','4','5','6','7','8','9'];
RESET(menu, '#4:menu.text');
FOR i := 0 TO 3 DO
    readln(menu, fdstfmenu[i]);
FOR i := 0 TO 4 DO
    readln(menu, fssmenu[i]);
FOR i := 0 TO 15 DO
    readln(menu, fssnewconmenu[i]);
fspnewconmenu[1] := fssnewconmenu[1];
fspnewconmenu[2] := fssnewconmenu[2];
fspnewconmenu[5] := fssnewconmenu[6];
fspnewconmenu[6] := fssnewconmenu[8];
fspnewconmenu[3] := 'Farmer name      ':';
fspnewconmenu[4] := 'Farmer address   ':';
FOR i := 0 TO 3 DO
BEGIN
    readln(menu, error1[i]);
    error2[i] := error1[i]
END;
error2[0] := 'Error ! No such contract exists';
FOR i := 0 TO 5 DO
    readln(menu, fssquerymenu[i]);
FOR i := 1 TO 6 DO
    readln(menu, fssshpmenu[i]);
FOR i := 0 TO 3 DO
    readln(menu, sciname[i]);
FOR i := 0 TO 4 DO
    readln(menu, fspmenu[i]);
FOR i := 0 TO 5 DO
    readln(menu, fspquerymenu[i]);
CLOSE(menu)
END; (* initialize *)

BEGIN (* main program *)
    initialize;
    IF password = ok      (* password is the boolean function *)
        THEN REPEAT
            prompt(fdstfmenu, 3, choice);
            CASE choice OF
                1 : fdstfsales;
                2 : fdstfpurchase;
                3 : quit := true
            END
        UNTIL quit

```

```
        ELSE write(at(30,25),chr(7),'Invalid password');  
END.
```

```
-----  
SEGMENT PROCEDURE fssnew;
```

```
(* to add new feed stuff contract and setup the data structure  
   accordingly *)
```

```
VAR  lineno,j,k,loc,choice,addr,i : integer;  
      goon,finish,done,locate : boolean;  
      temp : characters; ch:char;  
      tempdate : datatype;
```

```
PROCEDURE toconvert(line:characters;VAR tos:tostype);
```

```
CONST blank = ' ';
```

```
VAR temp : tostype;
```

```
      i,j,max,start : integer;
```

```
      item : characters;
```

```
BEGIN
```

```
  line := CONCAT(line,'$');
```

```
  max := LENGTH(line);
```

```
  WITH temp DO
```

```
    BEGIN
```

```
      month := 0; wgt := 0; bal := 0; unitprice := 0
```

```
    END;
```

```
    i := 1; j := 1;
```

```
    REPEAT
```

```
      WHILE (line[i] = blank) AND (i < max) DO
```

```
        i := i + 1;
```

```
      item := COPY(line,start,i-start);
```

```
      CASE j OF
```

```
        1 : temp.month := conint(item);
```

```
        2 : BEGIN
```

```
            temp.wgt := conint(item);
```

```
            temp.bal := temp.wgt
```

```
          END;
```

```
        3 : temp.unitprice := conreal(item)
```

```
      END; (* case *)
```

```
      j := j + 1
```

```
    UNTIL (j > 3) OR (i = max);
```

```
    tos := temp
```

```
END; (* toconvert *)
```

```
PROCEDURE getfssinfo;
```

```
(* get all the information for the new contract *)
```

```

FUNCTION proceed : boolean;
BEGIN
  IF EOF THEN BEGIN RESET(INPUT);EXIT(fssnew) END
  ELSE IF (lineno <> 8) AND (temp = ' ')
    THEN BEGIN
      proceed := false;
      lineno := lineno - 1;
    END
  ELSE IF (lineno = 8) AND (temp = ' ')
    THEN BEGIN
      proceed := false;
      k := k - 1
    END
  ELSE proceed := true;
END;

PROCEDURE tosinfo;
BEGIN
  WITH fsscontract DO
  BEGIN
    totalship := 0;
    write(at(x,9),' 8. ',fssnewconmenu[8]);
    finish := false; k := 0;
  REPEAT
    k := k + 1;
    GOTOXY(x+33,9+k); readln(temp);
    finish := (temp = 'F') OR (temp = 'f');
    IF (proceed) AND (NOT finish)
      THEN BEGIN
        tosconvert(temp,timeofship[k]);
        IF(timeofship[k].month < 1) OR
          (timeofship[k].month > 12)
          THEN BEGIN
            write(at(x+30,9+k),'Error in input,
              press <RETURN>');
            readln;
            write(at(x+30,9+k),' ':30);
            k := k - 1;
          END
        ELSE totalship := totalship + timeofship[k].wgt
      END;
  UNTIL (k=6) OR (finish);
  balofship := totalship;
  IF k < 6 THEN BEGIN
    write(at(x+30,9+k),' ':30);

```

```

timeofship[k].month := 0
(* 0 is endofdate marker *)
END

```

```

END
END;

```

```

PROCEDURE readnextinput;
BEGIN

```

```

    WITH fsscontract DO

```

```

        CASE lineno OF

```

```

            1,3,4,5,6,7 :

```

```

                BEGIN

```

```

                    write(at(x,lineno),lineno:2,' ',fssnewconmenu[lineno]);

```

```

                    readln(temp);

```

```

                    IF proceed

```

```

                        THEN CASE lineno OF

```

```

                            1 : number := temp;

```

```

                            3 : customer.name := temp;

```

```

                            4 : customer.contrno := temp;

```

```

                            5 : customer.addr := temp;

```

```

                            6 : commodity := temp;

```

```

                            7 : pricebase := temp;

```

```

                        END; (* case *)

```

```

                    END;

```

```

            2 : BEGIN

```

```

                write(at(x,2),' 2. ',fssnewconmenu[2]);

```

```

                readln(temp);

```

```

                IF proceed

```

```

                    THEN BEGIN

```

```

                        dateconvert(temp,contrdate);

```

```

                        IF datecheck(contrdate) <> ok

```

```

                            THEN BEGIN

```

```

                                write(at(38,2),'Error in input,
                                    press <RETURN>');

```

```

                                readln;

```

```

                                write(at(38,2),' ':30);

```

```

                                lineno := lineno - 1;

```

```

                                END

```

```

                            END

```

```

                    END;

```

```

            8 : tosinfo;

```

```

            10,11 :

```

```

                BEGIN

```

```

                    write(at(x,lineno+k+1),lineno:2,' ',

```

```

                                fssnewconmenu[lineno]);
readln(temp);
IF proceed
  THEN BEGIN
    dateconvert(temp, tempdate);
    IF datecheck(tempdate) <> ok
      THEN BEGIN
        write(at(38, line+k+1), 'Error in input,
          press <RETURN>');
        readln;
        write(at(38, line+k+1), ' ':30);
        lineno := lineno - 1;
        END
      ELSE BEGIN
        IF lineno = 10
          THEN lc.expdate := tempdate
          ELSE lc.shipdate := tempdate
        END
      END
    END
  END;
12 : BEGIN
  write(at(x, lineno+k+1), lineno:2, ' ', fssnewconmenu[lineno]);
  readln(temp);
  IF proceed THEN BEGIN
    lc.amount := conreal(temp);
    lc.bal := lc.amount
  END;
  END;
9, 13, 14, 15 :
  BEGIN
    write(at(x, lineno+k+1), lineno:2, ' ', fssnewconmenu[lineno]);
    readln(temp);
    IF proceed
      THEN CASE lineno OF
        9 : lc.number := temp;
        13 : issuebank := temp;
        14 : drawbank := temp;
        15 : mitino := temp;
        END
      END
    END
  END (* case *)
END; (* readnextinput *)

PROCEDURE fssmodify;
(* to modify the fsscontract input information *)

```

```

BEGIN
  REPEAT
    REPEAT
      goon := true;
      write(at(55,22), 'Which line to change:');
      readln(lineno); write(at(55,22), ' ':24);
      IF (lineno < 1) OR (lineno > 15)
        THEN BEGIN
          write (at(55,22),chr(7),'No such line!Press <RET>');
          readln;
          write(at(55,22), ' ':25);
          goon := false
        END
      UNTIL goon;
      (* now erase the line to be changed *)
      IF lineno <= 6
        THEN BEGIN
          write(at(38,lineno), ' ':40);
          GOTOXY(38,lineno)
        END
      ELSE IF lineno = 7
        THEN BEGIN
          write(at(38,lineno), ' ':80);
          GOTOXY(38,lineno)
        END
      ELSE IF lineno = 8
        THEN FOR j := 1 TO k DO
          write(at(43,9+j), ' ':20)
          (* no GOTOXY here since it is in PROC seven *)
        ELSE BEGIN
          write(at(38,lineno+k+1), ' ':40);
          GOTOXY(38,lineno+k+1)
        END;
      readnextinput;
      write(at(55,22), 'Ok now?(y/n)');
      read(ch);
    UNTIL (ch = 'Y') OR (ch = 'y')
  END;  (* fssmodify *)

```

```

BEGIN
  write(chr(12),at(15,0),fssnewconmenu[0]);
  lineno := 1;
  REPEAT
    readnextinput;

```

```

    lineno := lineno + 1;
UNTIL lineno > 15;
fsscontract.nofshipment := 0; fsscontract.status := occupied;
write(at(55,22),'Input OK?(y/n)');
read(ch); IF (ch = 'N') OR (ch = 'n') THEN fssmodify
END; (* getfssinfo *)

BEGIN (* fssnew *)
    getfssinfo; (*input all pertinent new sales contract info *)
    (* go thru the file and make sure that the given contract * is
       not already in the file *)
    RESET(fsshashfile,'*5:fsshashfile');
    REPEAT
        done := true;
        i := -1;
        REPEAT
            i := i + 1;
            SEEK(fsshashfile,i);
            GET(fsshashfile)
        UNTIL (EOF(fsshashfile)) OR
            (fsshashfile^.number = fsscontract.number);
        IF fsshashfile^.number = fsscontract.number
            THEN REPEAT (* error! same contract already in table *)
                prompt(error1,3,choice);
                CASE choice OF
                    1 : BEGIN CLOSE(fsshashfile);EXIT(fssnew) END;
                    2 : BEGIN CLOSE(fsshashfile);fssinquiry  END;
                    3 : BEGIN
                        REPEAT
                            write(chr(12),at(x,3),'Contract number
                                (<ctrl-c> to quit):');
                            readln(fsscontract.number);
                        UNTIL (fsscontract.number <> ' ') OR EOF;
                        done := false;
                        IF EOF THEN BEGIN
                            CLOSE(fsshashfile);
                            RESET(INPUT);
                            EXIT(fssnew)
                        END
                        END;
                END; (* case *)
            UNTIL choice = 3
        (* no error so put into the fssfile and fsshashfile *)
    ELSE BEGIN

```

```

(* place the new sales contract info into the fssfile;
   place at the first open slot *)
RESET(fssfile, '#5:fssfile');
(* put the contract info into the first open slot *)
loc := -1;
REPEAT
    loc := loc + 1;
    SEEK(fssfile, loc);
    GET(fssfile);
UNTIL (fssfile^.status = empty) OR (EOF(fssfile));
IF EOF(fssfile)
    THEN BEGIN
        write(chr(12), at(x,3), 'DOOMESDAY!No more space',
              ' to add new contract');
        write(at(x,4), 'Must use new diskette. Press<RET>');
        readln; CLOSE(fssfile); EXIT(fssnew)
    END;
fssfile^ := fsscontract;
SEEK (fssfile, loc);
PUT(fssfile); CLOSE(fssfile);
write(at(0,22), 'loc = ', loc);

(* find open slot in fsshashfile *)
RESET(fsshashfile);
i := -1;
REPEAT
    i := i + 1;
    SEEK(fsshashfile, i);
    GET(fsshashfile);
UNTIL fsshashfile^.status = empty;

(* put in the information *)
WITH fsshashfile^ DO
BEGIN
    status := occupied;
    number := fsscontract.number;
    name := fsscontract.customer.name;
    link := loc;
    commodity := fsscontract.commodity
END;
SEEK(fsshashfile, i);
PUT(fsshashfile);
CLOSE(fsshashfile)
END
UNTIL done

```

END; (* fssnew *)

SEGMENT PROCEDURE fssshipment;

VAR choice, loc, sfloc, i, j, m, old : integer;
done, bankpay, lastship : boolean;
inp : string[10];
bankpayamt : real;
ch : char;

PROCEDURE computepart;

BEGIN

WITH fsscontract DO

BEGIN

i := 1; (8 find out which months *)

write(chr(12), at(18, 0), 'month @price');

REPEAT

write(at(20, i), timeofship[i].month:2, ' \$',
timeofship[i].unitprice:7:2);

IF timeofship[i].month = fssshiprec.invoicedate.month
THEN BEGIN

m := i;

rate := timeofship[i].unitprice

END;

i := i + 1;

UNTIL (i > 6) OR (timeofship[i].month = 0);

write(at(20, i+2), 'Compute the price with the above rate?(y/n)');

read(ch);

IF ch IN ['N', 'n']

THEN REPEAT

write(at(20, 10), 'Rate = ');

readln(inp); IF inp <> ' ' THEN rate := conreal(inp)

UNTIL inp <> ' '

ELSE IF i > 6

THEN REPEAT

write(at(30, i+3), 'But you must give me the rate');

write(at(20, 10), 'Rate = ');

readln(inp); IF inp <> ' ' THEN rate := conreal(inp)

UNTIL inp <> ' ';

shpamtintons := fssshiprec.totalnet / 2000;

timeofship[m].bal := timeofship[m].bal - shpamtintons;

balofship := balofship - shpamtintons;

IF (timeofship[m].bal < 0) AND (timeofship[m+1].month <> 0)

THEN timeofship[m+1].bal := timeofship[m+1].bal

```

                                + timeofship[m].bal;
IF balofship < 0
    THEN lastship := true;
lc.bal := lc.bal - shpamtintons * rate;
IF lc.bal < 0
    THEN BEGIN
        bankpay := true;
        bankpayamt := -lc.bal
    END;
IF nofshipment <> 0
    THEN BEGIN                                (* more than one shipments so *)
        i := 0; j := shipmentinfo; (* link up the shipment records*)
        RESET(fssshipfile, '#5:fssshipfile');
        REPEAT
            SEEK(fssshipfile, j); old := j;
            GET(fssshipfile);
            j := fssshipfile^.link;
            i := i + 1;
        UNTIL i = nofshipment;
        fssshipfile^.link := sfloc;
        SEEK(fssshipfile, old); PUT(fssshipfile);
        CLOSE(fssshipfile); (* put back into the original place *)
    END
    ELSE shipmentinfo := sfloc; (* first shipment *)
nofshipment := nofshipment + 1;
END; (* with *)
END; (* computepart *)

BEGIN (* fssshipment *)
    RESET(fsshashfile, '#5:fsshashfile');
    REPEAT (* till the correct contract no given *)
        done := true;
        REPEAT
            write(chr(12), at(x, 3), 'Contract number: ');
            readln(contractno)
        UNTIL (contractno <> ' ') OR EOF;
    IF EOF THEN BEGIN RESET(INPUT);
                        CLOSE(fsshashfile); EXIT(fssshipment) END;

    i := -1;
    REPEAT
        i := i + 1;
        SEEK(fsshashfile, i);
        GET(fsshashfile)
    UNTIL (EOF(fsshashfile) OR (fsshashfile^.number = contractno));
    IF EOF(fsshashfile)

```

```

THEN REPEAT
    prompt(error2,3,choice);
    CASE choice OF
        1 : BEGIN CLOSE(fsshashfile);EXIT(fssshipment) END;
        2 : BEGIN CLOSE(fsshashfile);fssinquery END;
        3 : done := false
    END;
    UNTIL choice = 3
ELSE BEGIN (* ok find the contract info from fssfile *)
    loc := fsshashfile^.link; CLOSE(fsshashfile);
    RESET(fssfile,'#5:fssfile');
    SEEK(fssfile,loc);
    GET(fssfile); fsscontract := fssfile^;
    CLOSE(fssfile)
    END;
UNTIL done;

getshipinfo;
(* put this info into the fssshipfile *)
bankpay := false; lastship := false;
RESET(fssshipfile,'#5:fssshipfile'); sfloc := -1;
REPEAT
    sfloc := sfloc + 1;
    SEEK(fssshipfile,sfloc);
    GET(fssshipfile)
UNTIL (EOF(fssshipfile)) OR (fssshipfile^.status = empty);
write(at(0,22),'sfloc = ',sfloc);

IF EOF(fssshipfile)
THEN BEGIN
    CLOSE(fssshipfile);
    write(chr(12),chr(7),at(x,2),
        'DOOMESDAY no more space in the file');
    write(at(x,3),'Please call the system designer');
    write(at(x,4),'Meantime press <RETURN> and
        do other work');
    readln; EXIT(fssshipment)
    END;
fssshipfile^ := fssshiprec;
SEEK(fssshipfile,sfloc);
PUT(fssshipfile); CLOSE(fssshipfile);

(* make the necessary computation and save it *)
computepart;
(* now put it into the fssfile *)

```

```

RESET(fssfile, '#5:fssfile');
fssfile^ := fsscontract;
SEEK(fssfile, loc); PUT(fssfile);
CLOSE(fssfile);

```

```

shippaperwork
END; (* fssshipment *)

```

```

SEGMENT PROCEDURE listtoscreen;
VAR k : integer; twodecl : characters;

```

```

PROCEDURE listtop; (* half of contract information *)
BEGIN

```

```

    WITH fsscontract DO
    BEGIN
        write(chr(12), at(15, 0), 'Feed Stuff Sales Contract Information');
        write(at(x, 1), fssnewconmenu[1], ' ', number);
        write(at(x, 2), fssnewconmenu[2], ' ', contrdate.month, '/',
            contrdate.day, '/', contrdate.year);
        write(at(x, 3), fssnewconmenu[3], ' ', customer.name);
        write(at(x, 4), fssnewconmenu[4], ' ', customer.contrno);
        write(at(x, 5), fssnewconmenu[5], ' ', customer.addr);
        write(at(x, 6), fssnewconmenu[6], ' ', commodity);
        write(at(x, 7), fssnewconmenu[7], ' ', pricebase);
    END;

```

```

END; (* listtop *)

```

```

PROCEDURE listbottom;
(* bottom half of fsscontract info *)

```

```

BEGIN
    WITH fsscontract DO
    BEGIN
        k := 1;
        write(at(x, 9),
            'Time of shipment : Months Quantity Balance Unitprice');
        REPEAT
            write(at(29+x, 9+x), timeofship[k].month:2, timeofship[k].wgt:10,
                timeofship[k].bal:12:2, ' $', timeofship[k].unitprice:8:2);
            k := k + 1;
        UNTIL (timeofship[k].month = 0) OR (k > 6);
        write(at(x, 9+k), 'total shipment : ', totalship:8:2);
        write(at(x, 10+k), 'balance of shipment : ', balofship:8:2);
        write(at(0, 22), 'Press <RETURN>'); readln;
        write(at(x, 2), fssnewconmenu[10], ' ', lc.expdate.month, '/',
            lc.expdate.day, '/', lc.expdate.year);
        write(at(x, 3), fssnewconmenu[11], ' ', lc.shipdate.month, '/',

```

```

        lc.shipdate.day, '/', lc.shipdate.year);
dollarcent(lc.amount, twodeci); addcomma(twodeci);
write(at(x, 4), fssnewconmenu[12], ' $', twodeci:10);
dollarcent(lc.bal, twodeci); addcomma(twodeci);
write(at(x, 5), 'L/C balance           : $', twodeci:10);
write(at(x, 6), fssnewconmenu[13], issuebank);
write(at(x, 7), fssnewconmenu[14], drawbank);
write(at(x, 8), faanewconmenu[15], ' ', mitino);
write(at(x, 9), '* of shipment made : ', nofshipment);
END; (* with *)
END; (* listbottom *)

```

```

BEGIN {listtoscreen}
    listtop;
    listbottom
END;

```

```

SEGMENT PROCEDURE listtopprinter;
VAR i, k : integer; str1, str2, twodeci : characters;

```

```

PROCEDURE prlisttop; (* half of contract info to printer *)
BEGIN
    WITH fsscontract DO
    BEGIN
        skip(4);
        writeln(out, ' ':20, 'Feed Stuff Sales Contract Information');
        skip(3);
        writeln(out, ' ':10, fssnewconmenu[1], ' ', number); writeln(out);
        writeln(out, ' ':10, fssnewconmenu[2], ' ', contrdate.month, '/',
            contrdate.day, '/', contrdate.year); writeln(out);
        writeln(out, ' ':10, fssnewconmenu[3], ' ', customer.name);
        writeln(out);
        writeln(out, ' ':10, fssnewconmenu[4], ' ', customer.contrno);
        writeln(out);
        writeln(out, ' ':10, fssnewconmenu[5], ' ', customer.addr);
        writeln(out);
        writeln(out, ' ':10, fssnewconmenu[6], ' ', commodity);
        writeln(out);
        IF LENGTH(pricebase) < 41
            THEN writeln(out, ' ':10, fssnewconmenu[7], ' ', pricebase)
            ELSE BEGIN
                i := 41;
                REPEAT
                    i := i - 1

```

```

        UNTIL pricebase[i] = ' ';
        str1 := COPY(pricebase,1,i-1);
        str2 := COPY(pricebase,i+1,LENGTH(pricebase)-i);
        writeln(out, ' ':10,fssnewconmenu[7], ' ',str1);
        writeln(out, ' ':35,str2)
        END;
    END;
END; (* prlisttop *)

PROCEDURE prlistbottom;
(* bottom half of fsscontract info to the printer *)
BEGIN
    WITH fsscontract DO
    BEGIN
        skip(2); k := 1;
        writeln(out, ' ':10,
        'Time of shipment      : Months Quantity Balance Unitprice');
        REPEAT
            writeln(out, ' ':39,timeofship[k].month:2,timeofship[k].wgt:10,
            timeofship[k].bal:12:2, ' $', timsofship[k].unitprice:8:2);
            k := k + 1;
        UNTIL (timeofship[k].month = 0) OR (k > 6); skip(2);
        writeln(out, ' ':10,'total shipment      : ',totalship:8:2);
        writeln(out);
        writeln(out, ' ':10,'balance of shipment : ',balofship:8:2);
        writeln(out);
        writeln(out, ' ':10,fssnewconmenu[9], ' ',lc.number);writeln(out);
        writeln(out, ' ':10,fssnewconmenu[10], ' ',lc.expdate.month,'/',
            lc.expdate.day,'/',lc.expdate.year); writeln(out);
        writeln(out, ' ':10,fssnewconmenu[11], ' ',lc.shipdate.month,'/',
            lc.shipdate.day,'/',lc.shipdate.year); writeln(out);
        dollarcent(lc.amount,twodeci);addcomma(twodeci);
        writeln(out, ' ':10,fssnewconmenu[12], ' $',twodeci:10);
        writeln(out);
        dollarcent(lc.bal,twodeci); addcomma(twodeci);
        writeln(out, ' ':10,'L/C balance          :$ ', twodeci:10);
        writeln(out);
        writeln(out, ' ':10,fssnewconmenu[13], ' ',issuebank);writeln(out);
        writeln(out, ' ':10,fssnewconmenu[14], ' ',drawbank);writeln(out);
        writeln(out, ' ':10,fssnewconmenu[15], ' ',mitino);writeln(out);
        writeln(out, ' ':10,'* of shipment made   : ',nofshipment);
    END; (* with *)
END; (* prlistbottom *)

BEGIN {listtoprinter}

```

```

prlisttop;
prlistbottom
END;

```

```

SEGMENT PROCEDURE fssresiduecheck;

```

```

VAR j,opencnt : integer;

```

```

BEGIN

```

```

    write(chr(12),at(x,2),'Available Space'); opencnt := 0;

```

```

    j := 0; RESET(fssfile,'*5:fssfile');

```

```

    SEEK(fssfile);

```

```

    GET(fssfile);

```

```

    write(at(x,4),'Contract file: ');

```

```

    WHILE NOT EOF(fssfile) DO

```

```

    BEGIN

```

```

        IF fssfile^.status = empty

```

```

        THEN BEGIN

```

```

            opencnt := opencnt + 1;

```

```

            write(at(x+19,4),opencnt:3)

```

```

        END;

```

```

        j := j + 1;

```

```

        SEEK(fssfile,j);

```

```

        GET(fssfile)

```

```

    END;

```

```

    CLOSE(fssfile);

```

```

    opencnt := 0; j := 0;

```

```

    write(at(x,5),'Shipment file: ');

```

```

    RESET(fssshipfile,'*5:fssshipfile');

```

```

    SEEK(fssshipfile,j); GET(fssshipfile);

```

```

    WHILE NOT EOF(fssshipfile) DO

```

```

    BEGIN

```

```

        IF fssshipfile^.status = empty

```

```

        THEN BEGIN

```

```

            opencnt := opencnt + 1;

```

```

            write(at(x+19,5),opencnt:3)

```

```

        END;

```

```

        j := j + 1;

```

```

        SEEK(fssshipfile,j);

```

```

        GET(fssshipfile)

```

```

    END;

```

```

    CLOSE(fssshipfile);

```

```

    write(at(0,7),'Press <RETURN>'); readln

```

```

END; {fssresiduecheck}

```

- 4

```

SEGMENT PROCEDURE fssinquery;

```

```

VAR i,j,k,num,entries,loc : integer;
    compname : string[25];
    quit : boolean;
    customlist : intype; ch : char; twodeci, str1, str2: characters;

PROCEDURE listship;
VAR sumbales, sumnet : real;
BEGIN
    WITH fssshiprec DO
    BEGIN
        write(chr(12), at(0,0), 'Shipname : ', name);
        write(at(38,0), 'inv date: ', invoicedate.month, '/',
            invoicedate.day, '/', invoicedate.year);
        write(at(60,0), 'etd: ', etd.month, '/', etd.day, '/', etd.year);
        write(at(0,1), 'origin port: ', origin);
        write(at(38,1), 'destination port: ', dest);
        write(at(18,2), 'Container#', at(33,2), 'Bales', at(41,2), 'Net wgt');
        sumbales := 0; sumnet := 0;
        FOR k := 1 TO nofcont DO
            WITH container[k] DO
            BEGIN
                sumbales := sumbales + bales;
                sumnet := sumnet + net;
                prealtostr(net, twodeci); addcomma(twodeci0;
                write(at(18,3+k), number, at(31,3+k), bales:5, at(40,3+k),
                    twodeci:7)
            END;
            prealtostr(sumbales, str1); addcomma(str1);
            prealtostr(sumnet, str2); addcomma(str2);
            writeln(at(31, nofcont+4), '-----', at(40, nofcont+4), '-----');
            writeln(at(26, nofcont+5), str1:10, str2:11);
        END
    END;
END;

```

```

PROCEDURE prlistship;
VAR sumbales, sumnet : real;
BEGIN
    WITH fssshiprec DO
    BEGIN
        write(at(0,20), ' ':70);
        write(at(0,22), 'Need a printout?(y/n)');
        read(ch); IF EOF THEN BEGIN RESET(INPUT); EXIT(prlistship) END;
        IF ch IN ['Y', 'y']
            THEN BEGIN

```

```

write(at(0,22),'Turn on the TEC and press <RETURN>');
readln;
REWRITE(out,'PRINTER:');skip(4);
writeln(out,'  Shipment No.',num,
  '      Invoice No.',fssshiprec.invoiceno);writeln(out);
writeln(out,'  shipname: ',name);skip(2);
writeln(out,'  invoice date: ',invoicedate.month,'/',
  invoicedate.day,'/',invoicedate.year);writeln(out);
writeln(out,'  etd      :',etd.month,'/',etd.day,'/',
  etd.year);skip(3);
writeln(out,'  :18,Container #',' :5,'Bales',' :5,
  'Net wgt');skip(2);
sumbales := 0; sumnet := 0;
FOR k := 1 TO nofcont DO
  WITH container[k] DO
    BEGIN
      sumbales := sumbales + bales;
      sumnet := sumnet + net;
      prealtostr(net,twodeci);addcomma(twodeci);
      writeln(out,' :18,number:12,' :5,bales:5,' :5,
        twodeci:7);
      writeln(out);
    END;
    prealtostr(sumbales,str1);addcomma(str1);
    prealtostr(sumnet,str2); addcomma(str2);
    writeln(out,' :35,'-----',' :5,'-----');
    writeln(out,' :30,str1:10,str2:12);
  CLOSE(out)
END
END; (* with *)
END;

PROCEDURE onecontrinfo;

PROCEDURE case3sub;
BEGIN
  write(at(0,22),'Need a printout?(y/n)');read(ch);
  IF EOF THEN BEGIN RESET(INPUT);EXIT(case3sub) END;
  IF ch IN ['Y','y']
    THEN BEGIN
      write(at(0,22),'Turn on the TEC and press <RETURN>');
      readln; REWRITE(out,'printer:');
      listtoprinter; CLOSE (out)
    END;
  IF fsscontract.nofshipment > 0 THEN

```

```

BEGIN
write(at(0,22),
  'Like to see all shipments in sequence?(y/n/<ctrl-c> to quit)');
read(ch); IF EOF THEN BEGIN RESET(INPUT);EXIT(case3sub) END;
IF ch IN ['Y','y']
THEN BEGIN
  i := 0; j := fsscontract.shipmentinfo;
  RESET(fssshipfile, '#5:fssshipfile');
  REPEAT
    SEEK(fssshipfile);
    GET(fssshipfile);
    j := fssshipfile^.link; i := i + 1; num := i;
    fssshiprec := fssshipfile^;
    listship; prlistship;
    IF i < fsscontract.nofshipment
    THEN BEGIN
      write(at(0,22),
        'Want to see next shipment?(y/n) ');
      read(ch); write(at(0,22), ' ':50);
      IF EOF THEN BEGIN CLOSE(fssshipfile);
      RESET(INPUT);EXIT(case3sub) END
      END
    UNTIL (i = fsscontract.nofshipment) OR (ch IN
      ['N','n']);
    CLOSE(fssshipfile);
  END
ELSE BEGIN
  REPEAT
    write(at(0,22), ' ':50);    (* erase previous line *)
    REPEAT
      write(at(0,22), 'Which shipment(<ctrl-c> to quit):');
      readln(num); IF EOF THEN BEGIN RESET(INPUT);
      EXIT(case3sub) END
    UNTIL (num >= 1) AND
      (num <= fsscontract.nofshipment);
    (* locate the shipment info *)
    i := 0; j := fsscontract.shipmentinfo;
    RESET(fssshipfile, '#5:fssshipfile');
    REPEAT
      SEEK(fssshipfile, j);
      GET(fssshipfile);
      j := fssshipfile^.link; i := i + 1;
    UNTIL i = num;
    fssshiprec := fssshipfile^; CLOSE(fssshipfile);
    listship; prlistship;

```

```

        write(at(0,22),'Like to see another shipment?(y/n)');
        read(ch); IF EOF THEN BEGIN RESET(INPUT);
        EXIT(case3sub) END
    UNTIL ch IN ['N','n']
    END (* else *)
END
END; (* case3sub *)

BEGIN (* onecontrinfo *)
    REPEAT
        write(chr(12),at(x,3),'Contract number:');
        readln(contractno);
    UNTIL (contractno <> ' ') OR EOF; (* not empty or terminate *)
    IF EOF THEN BEGIN RESET(INPUT);EXIT(fssinquiry) END;
    RESET(fsshashfile,'#5:fsshashfile'); i := -1;
    REPEAT
        i := i + 1;
        SEEK(fsshashfile,i);
        GET(fsshashfile)
    UNTIL (EOF(fsshashfile)) OR (fsshashfile^.number=contractno);
    IF EOF(fsshashfile) (* not found *)
    THEN BEGIN
        CLOSE(fsshashfile);
        write(at(x,5),chr(7),'No such contract in the file');
        write(at(x,7),'Press <RETURN>'); readln;
        END
    ELSE BEGIN
        CLOSE(fsshashfile);
        loc := fsshashfile^.link;
        RESET(fssfile,'#5:fssfile');
        SEEK(fssfile,loc); GET(fssfile); CLOSE(fssfile);
        fsscontract := fssfile^;
        listtoscreen;
        case3sub
    END;
END;

BEGIN (* fssinquiry *)
    quit := false;
    REPEAT
        prompt(fssquerrymenu,5,choice);
        CASE choice OF
            1 : BEGIN
                RESET(fsshashfile,'#5:fsshashfile');
                write(chr(12),at(15,0),'List of all customer');

```

```

entries := 1; customlist[1] := ' '; j := -1;
REPEAT
  REPEAT
    j := j + 1;
    SEEK(fsshashfile,j);
    GET(fsshashfile)
  UNTIL(EOF(fsshashfile)) OR
    (fsshashfile^.status <> empty);
  IF NOT EOF(fsshashfile)
    THEN BEGIN
      i := 1;
      (* check if this record's name is already in
         customers array, if not put it *)
      while (i<entries) AND (fsshashfile^.name
        <> customlist[i]) DO
        i := i + 1;
      IF i = entries
        (* not in customers array so put it in *)
        THEN BEGIN
          customlist[i] := fsshashfile^.name;
          entries := entries + 1
        END;
      END
    UNTIL EOF(fsshashfile);
  CLOSE(fsshashfile); i := 1;
  WHILE i <= entries - 1 DO
  BEGIN
    IF validate(customlist[i])
      THEN write(at(x,i),customlist[i]);
    i := i + 1
  END;
  write(at(55,22),'Press <RETURN>');readln
END;
2 : BEGIN
  REPEAT
    write(chr(12),at(x,3),'Company name:');
    readln(compname)
  UNTIL (compname <> ' ') OR EOF;
  (*not empty or terminate *)
  IF EOF THEN BEGIN RESET(INPUT);EXIT(fssinquiry)
    END;
  RESET (fsshashfile,'#5:fsshashfile');
  write(chr(12),at(15,0),'Contract with ',compname);
  j := 2;
  FOR i := 0 TO max DO

```

```

BEGIN
    SEEK(fsshashfile,1);
    GET(fsshashfile);
    IF (fsshashfile^.name = occupied)
        AND (fsshashfile^.name = compname)
    THEN BEGIN
        write(at(x,j),fsshashfile^.number,' ',
            fsshashfile^.commodity);
        j := j + 1;
    END;
    IF j = 22 (* full screen *)
    THEN BEGIN
        write(at(55,22),'Press <RETURN>');
        readln;write(chr(12)); j := 2
    END;
END;
write(at(55,22),'Press <RETURN>');readln;
CLOSE(fsshashfile)
END;
3 : onecontrinfo;
4 : fssresiduecheck;
5 : quit := true;
END; (* case *)
UNTIL quit
END;

```

```

SEGMENT PROCEDURE getshipinfo;
VAR k,lineno,xaxis : integer;
    ch : char;
    finish,goon : boolean;
    temp : characters;
    tempdate : datatype;

PROCEDURE contconvert(line:characters;VAR cont:contype);
CONST blank = ' ';
VAR temp : contype;
    i,j,max,start : integer;
    gross,tare : real;
    item : characters;
BEGIN
    line := CONCAT(line,'$');
    WITH temp DO
    BEGIN
        number := blank; net := 0; bales := 0;
    END;

```

```

temp.number := COPY(line,1,12);
DELETE(line,1,12);
i := 1; j := 1; max := LENGTH(line);
REPEAT
    WHILE (line[i] = blank) AND (i<max) DO
        i := i + 1;
    start := i;
    WHILE (line[i] <> blank) AND (i < max) DO
        i := i + 1;
    item := COPY(line,start,i-start);
    CASE j OF
        1 : temp.bales := conint(item);
        2 : gross      := conreal(item);
        3 : BEGIN
            tare := conreal(item);
            temp.net := gross - tare
        END
    END; (* case *)
    j := j + 1
UNTIL (j > 3) OR (i = max);
cont := temp
END;

FUNCTION sproceed : boolean;
BEGIN
    IF EOF THEN BEGIN RESET(INPUT);EXIT(getshipinfo) END
    ELSE IF temp = ' '
        THEN BEGIN
            sproceed := false;
            lineno := lineno -1
        END
    ELSE sproceed := true
END;

PROCEDURE nextshipinput;
BEGIN
    WITH fssshiprec DO
    BEGIN
        CASE lineno OF
            1 : BEGIN
                write(at(0,0),'1.',fssshpmenu[1]);
                readln(temp);
                IF sproceed THEN name := temp
            END;
            2,3 : BEGIN

```

```

IF lineno = 2 THEN xaxis := 38 ELSE xaxis := 60;
write(at(xaxis,0),lineno,'.',fssshpmenu[lineno]);
readln(temp);
IF sproceed
  THEN BEGIN
    dateconvert(temp,tempdate);
    IF datecheck(tempdate) <> ok
      THEN BEGIN
        write(at(xaxis,0),'Error,press <RETURN>');
        readln;
        write(at(xaxis,0),' ':20);
        lineno := lineno - 1
      END
    ELSE IF lineno = 2
      THEN invoicedate := tempdate
      ELSE etd := tempdate
    END
  END;
4,5 : BEGIN
  IF lineno = 4 THEN xaxis := 0 ELSE xaxis := 38;
  write(at(xaxis,1),lineno,'.',fssshomenu[lineno]);
  readln(temp);
  IF sproceed
    THEN IF lineno = 4
      THEN origin := temp
      ELSE dest := temp
    END;
  END; (* case *)
IF (lineno >= 6)
  THEN BEGIN
    write(at(0,2),' ',fssshpmenu[6]);
    write(at(14,lineno-3),lineno:2,' ');
    readln(temp);
    finish := (temp = 'F') OR (temp = 'f');
    IF (sproceed) AND (NOT finish)
      THEN BEGIN
        contconvert(temp,container[lineno-5]);
        write(at(56,lineno-3),container[lineno-5].net:8:1);
        totalbales := totalbales+container[lineno-5].bales;
        totalnet := totalnet+container[lineno-5].net;
        IF lineno-5 > nofcont THEN nofcont := lineno-5
          {necessary not to reset nofcont when called from
            shipmodify}
        END;
      IF finish

```

```

        THEN write(at(14,lineno-3),' ':60)
    END {if}
END {with}
END; (* nextshipinput *)

PROCEDURE shipmodify;
BEGIN
    REPEAT
        REPEAT
            goon := true;
            write(at(55,22),'Which line to change:');
            readln(lineno); write(at(55,22),' ':24);
            IF (lineno < 1) OR (lineno > fssshiprec.nofcont + 5)
                THEN BEGIN
                    write(at(55,22),chr(7),'No such line,
                                press <RETURN>');
                    readln; write(at(55,22),' ':25); goon := false
                END
        UNTIL goon;
        (* now erase the line to be changed *)
        IF lineno <= 5
            THEN CASE lineno OF
                1 : write(at(11,0),' ':25);
                2 : write(at(49,0),' ':11);
                3 : write(at(66,0),' ':12);
                4 : write(at(14,1),' ':24);
                5 : write(at(57,1),' ':23);
            END (* case *)
        ELSE WITH fssshiprec DO
            BEGIN
                write(at(18,lineno-3),' ':60);
                totalbales := totalbales - container[lineno-5].bales;
                totalnet := totalnet - container[lineno-5].net
            END;
            (* read the modifying line *)
            nextshipinput;
            write(at(55,22),'Ok now?(y/n)'); read(ch)
        UNTIL ch IN ['Y','y']
    END; (* shipmodify *)

BEGIN (* getshipinfo *)
    lineno := 1; fssshiprec.totalbales := 0;
    fssshiprec.totalnet := 0; fssshiprec.nofcont := 0;
    REPEAT { get the invoice number }
        write(chr(12),at(x,3),'Invoice number(append):',

```

```

                                fsscontract.number);
readln(fssshiprec.invoiceno);
fssshiprec.invoiceno:=CONCAT(fsscontract.number,
                                fssshiprec.invoiceno);
write(at(x,5),'Invoice no. will be ',fssshiprec.invoiceno);
write(at(x,6),'Correct?(y/n)');
read(ch)
UNTIL ch IN ['Y','y']; writeln(chr(12)); finish := false;
REPEAT
    nextshipinput;
    lineno := lineno + 1
UNTIL (lineno > 25) OR (finish);
fssshiprec.status := occupied;
write(at(55,22),'Input OK?(y/n)'); read(ch);
IF ch IN ['N','n'] THEN shipmodify
END;
```

```

SEGMENT PROCEDURE shippaperwork;
VAR calendar : ARRAY[1..12] OF string[9];
    format1,format2,format3,firsthalf,sechalf:characters;
    k,choice,i,oneline,casecnt : integer;
    ch : char;
```

```

PROCEDURE signature;
BEGIN
    skip(3);
    writeln(out,' ':20,'KOBE    MERCHANTILE , INC');writeln(out);
    writeln(out,' ':20,'signed _____');
    writeln(out,' ':20,'S.HANAOKA,General Manager');
END;
```

```

PROCEDURE underline(i,j : integer);
BEGIN
    write(out,' ':i);
    FOR i := 1 TO j DO
        write(out,'-');
    writeln(out)
END;
```

```

PROCEDURE ashipinv;
BEGIN
    WITH fsscontract,fssshiprec DO
    BEGIN
```

```

skip(8);
writeln(out, ' ':65,calendar[invoicedate.month], ' ',
    invoicedate.day, ',19',invoicedate.year);writeln(out);
writeln(out, ' ':7,invoiceno, ' ':24-LENGTH(invoiceno),
    shpamtintons:7:3, ' shorttons of',commodity);
writeln(out);
writeln(out, ' ':43,name);writeln(out);
writeln(out, ' ':25,calendar[etd.month], ' ',etd.day, ',19',
    etd.year); writeln(out);
writeln(out, ' ':9,origin, ' ':36-LENGTH(origin),dest);
writeln(out);
writeln(out, ' ':26,customer.name);writeln(out);
writeln(out, ' ':8,customer.addr);writeln(out);
writeln(out, ' ':19,customer.contrno,
    ' ':41-LENGTH(customer.contrno),lc.number);
writeln(out);
writeln(out, ' ':19,number, ' ':41-LENGTH(number),mitino);
skip(5);
writeln(out, ' ':18,commodity);
END
END;

PROCEDURE bshipinv;
BEGIN
    WITH fssshiprec,fsscontract DO
    BEGIN
        format2 := pricebase;
        WHILE LENGTH(format2) > 31 DO
        { write pricebase using 2 or 3 lines }
        BEGIN
            i := 31;
            REPEAT  i := i + 1  UNTIL format2[i] = ' ';
            format1 := COPY(format2,1,i-1);
            format2 := COPY(format2,i+1,LENGTH(format2)-i);
            writeln(out, ' ':51,format1)
        END;
        writeln(out, ' ':51,format2);
        writeln(out, ' ':17,'Container#    Bales    Net wgt(lbs)');
        writeln(out, ' ':17,'-----    -----    -----');
        FOR k := 1 TO nofcont DO
        WITH container[k] DO
        BEGIN
            prealtostr(net,format1);addcomma(format1);
            writeln(out, ' ':17,number, ' ':14-LENGTH(number),bales:4,
                ' ':3,format1:10)
        END
        END
    END
END

```

```

END;
writeln(out, ' ':17, '_____');
writeln(out, ' ':3, nofcont, ' X 40 foot');
writeln(out, ' ':3, 'CONTAINERS');
prealtostr(totalnet, format1); addcomma(format10;
dollarcent(shpamtintons*rate, format2); addcomma(format2);
STR(totalbales, format3); addcomma(format3);
writeln(out, ' ':17, 'Total', format3:6, ' Bales ', format1:10,
        ' lbs');
writeln(out, ' ':17, format1:9, ' lbs=', shpamtintons:7:3,
        ' shorttons', ' @US', rate:8:2, ' US$', format2:10);
writeln(out, ' ':17, '=====',
        ' ':14, '====='); signature
END
END;

PROCEDURE ashippaklist;
BEGIN
  WITH fsscontract, fssshiprec DO
  BEGIN
    skip(9);
    writeln(out, ' ':30, 'P A C K I N G   L I S T ');
    writeln(out, ' ':30, '-----'); skip(2);
    underline(3,74);
    writeln(out, ' ':3, 'INVOICE NO: ', invoiceno,
      ' ':29-LENGTH(invoiceno), 'DATE: ',
      calendar[invoicedate.month], ' ', invoicedate.day,
      ',19', invoicedate.year); underline(4,72);
    writeln(out, ' ':3, 'MESSRS: ', customer.name);
    writeln(out, ' ':11, customer.addr); skip(2); underline(4,72);
    writeln(out, ' ':3, 'SHIPPED PER: ', name, ' ':28-LENGTH(name),
      'SAILING ON/OR ABOUT: 'calendar[etd.month], ' ',
      etd.day, ',19', etd.year); underline(3,74);
    writeln(out, ' ':3, 'FROM: ', origin, ' ':35-LENGTH(origin), 'TO: ',
      dest); underline(3,74);
    writeln(out, ' ':3, 'MARKS & NOS. ', ' ':29, 'DESCRIPTION');
    writeln(out, ' ':3, '-----', ' ':29, '-----');
    skip(2);
    writeln(out, ' ':44, commodity); writeln(out);
    writeln(out, ' ':33, 'Container#      Bales      Net Weight(lbs)');
    writeln(out, ' ':33, '-----      -----      -----');
    writeln(out);
  END
END;

```

```

PROCEDURE bshippaklist;
BEGIN
  WITH fsscontract, fssshiprec DO
  BEGIN
    FOR k := 1 TO nofcont DO
      WITH container[k] DO
      BEGIN
        prealtostr(net, format1); addcomma(format1);
        writeln(out, ' ':33, number, ' ':15-LENGTH(number), bales:6,
          ' ':5, format1:10)
      END;
      writeln(out, ' ':33, '-----');
      write(out, ' ':3, nofcont, ' X 40 FOOT CONTAINERS');
      STR(totalbales, format1); addcomma(format1);
      prealtostr(totalnet, format2); addcomma(format2);
      writeln(out, ' ':9, 'Total', format1:9, ' Bales', format2:13, ' lbs');
      writeln(out, ' ':33, format2:10, ' lbs =', shpamtintons:10:3,
        ' shorttons');
      writeln(out, ' ':33, '=====');
      signature
    END
  END;
END;

```

```

PROCEDURE acertorigin;
BEGIN
  WITH fsscontract, fssshiprec DO
  BEGIN
    skip(6); writeln(out, ' ':12, 'Shunsuke Hanaoka'); writeln(out);
    writeln(out, ' Kobe Merchantile, Inc., 861 Six Ave., ',
      ' San Diego, CA. 92101');
    writeln(out); writeln(out, ' ':30, 'M / ', name); writeln(out);
    write(out, ' ':9, calendar[etd.month], ' ', etd.day, ' ', 19,
      etd.year, ' ':11, customer.name);
    IF LENGTH(customer.addr) <= 35
      THEN BEGIN
        writeln(out, ' ', customer.addr);
        skip(5)
      END
    ELSE BEGIN
      i := 35;
      REPEAT i := i - 1 UNTIL customer.addr[i] = ' ';
      firsthalf := COPY(customer.addr, 1, i-1);
      sechalf := COPY(customer.addr, i+1,
        LENGTH(customer.addr)- i);
      writeln(out, ' ', firsthalf); writeln(out);
    END
  END;
END;

```

```

        writeln(out, ' ', sechalf);
        skip(3)
    END;

END
END;

PROCEDURE bcertorigin;
BEGIN
    WITH fsscontract, fssshiprec DO
    BEGIN
        writeln(out, ' ':46, commodity); skip(3);
        FOR k := 1 TO nofcont DO
            writeln(out, ' ':4, container[k].number);
            skip(2);
            writeln(out, ' ':18, nofcont, ' X 40 FOOT');
            prealtostr(totalnet, format2); addcomma(format2);
            writeln(out, ' ':18, 'CONTAINERS', ' ':7, format2, ' lbs');
            skip(2);
            writeln(out, ' ':15, 'THESE COMMODITIES LICENSED BY THE U.S.'
                , ' FOR ULTIMATE DESTINATION');
            writeln(out, ' ':15, 'JAPAN. DIVERSION CONTRARY TO U.S.
                LAW PROHIBITED');
            skip(26-nofcont);
            writeln(out, ' ':5, 'Chamber of Commerce of San Diego');
            writeln(out, ' ':41, 'California')
        END
    END; (* bcertorigin *)

PROCEDURE aphytocert;
BEGIN
    WITH fsscontract, fssshiprec DO
    BEGIN
        prompt(sciname, 3, choice); write(chr(12));
        skip(9); writeln(out, ' ':33, 'JAPAN'); skip(18);
        writeln(out, ' ':28, 'Kobe Merchantile, Inc., 861 Six Ave.,
            San Diego, CA 92101');
        writeln(out, ' ':28, customer.name, ', ');
        writeln(out, ' ':28, customer.addr); writeln(out);
        writeln(out, ' ':38, shpamtintons:7:3, ' shorttons of ');
        writeln(out);
        writeln(out, ' ':38, commodity); writeln(out);
        writeln(out, ' ':38, sciname[choice]); writeln(out);
        writeln(out, ' ':30, nofcont, ' X 40 Foot Containers');
        IF nofcont <= 5 THEN writeln(out); l := 0; oneline := 0;
        REPEAT

```

```

write(out, ' ':20);
oneline := oneline + 5;
REPEAT
    i := i + 1;
    write(out, container[i].number:LENGTH(container[i].
                                                number), ',')
UNTIL (i = nofcont - 1) OR (i = oneline);
IF i = nofcont - 1
    THEN writeln(out, container[nofcont].number)
    ELSE writeln(out)
UNTIL i = nofcont - 1;
IF nofcont <= 10 THEN writeln(out);
END
END;

PROCEDURE bphytocert;
BEGIN
    WITH fsscontract, fssshiprec DO
    BEGIN
        writeln(out, ' ':14, 'Imperial County, California');
        writeln(out);
        writeln(out, ' ':21, 'Ocean Vessel', ' ':25, 'Japan');
        skip(3);
        write(out, ' ':15, 'This ', commodity, ', ', sciname[choice]);
        FOR i := 1 TO LENGTH(sciname[choice]) DO
            write(out, chr(8));      { backspace }
        FOR i := 1 TO LENGTH(sciname[choice]) DO
            write(out, '_');          { underline } skip(2);
        writeln(out, ' ':15, 'was grown in the Imperial County,
                                California. The Hessian Fly, ');
        writeln(out);
        write(out, ' ':15, 'Phytophaga Destructor (Say)');
        FOR i := 1 TO 27 DO
            write(out, chr(8));
        FOR i := 1 TO 27 DO
            write(out, '_');
        writeln(out, ' is not known to occur in'); writeln(out);
        writeln(out, ' ':15, 'the Imperial County, California. ');
    END { with }
END;

PROCEDURE fumigation;
BEGIN
    WITH fssshiprec DO
    BEGIN

```

```

skip(28);
IF nofcont <= 10
  THEN FOR k := 1 TO nofcont DO
    writeln(out, ' ':15, container[k].number)
  ELSE FOR k:= 1 TO 10 DO
    BEGIN
      write(out, ' ':15, container[k].number);
      IF k+10 <= nofcont
        THEN writeln(out, ' ':10, container[k+10].number)
        ELSE writeln(out)
    END;
  IF nofcont <= 10 THEN skip(13-nofcont)
    ELSE skip(3);
i := 0;
{ separate the shipname and voyage number }
REPEAT
  i := i + 1
UNTIL (name[i]='V') OR (name[i]='v') OR (i=LENGTH(name));
IF (name[i] <> 'v') AND (name[i] <> 'V')
  THEN BEGIN
    REPEAT { get the voy* since not given }
      write(chr(12), 'No voyage number, voyage #:');
      readln(format2)
    UNTIL format2 <> ' '; format1 := name;
    write(chr(12))
  END
  ELSE BEGIN
    format1 := COPY(name, 1, i-1);
    format2 := COPY(name, i, LENGTH(name)-i+1)
  END;
writeln(out, ' ':40, format1); writeln(out);
writeln(out, ' ':21, format2, ' ':28, dest)
END { with }
END;

BEGIN
  (* initialize *)
  calendar[1] := 'January';   calendar[2] := 'February';
  calendar[3] := 'March';     calendar[4] := 'April';
  calendar[5] := 'May';       calendar[6] := 'June';
  calendar[7] := 'July';      calendar[8] := 'August';
  calendar[9] := 'September'; calendar[10] := 'October';
  calendar[11] := 'November'; calendar[12] := 'December';

  REWRITE(out, 'PRINTER:');

```

```

casecnt := 0;
REPEAT
  casecnt := casecnt + 1;
  CASE casecnt OF
    1 : BEGIN
      write(chr(12),at(0,3),
        'Turn on the TEC, insert the INVOICE sheet and press
                                     <RETURN>');
      readln;
      ashipinv;
      bshipinv
    END;
    2 : BEGIN
      write(at(0,3),
        'Now put the PACKING LIST sheet and press <RETURN> ');
      readln;
      ashippaklist;
      bshippaklist
    END;
    3 : BEGIN
      write(at(0,3),
        'Certificate of Origin sheet, Press <RETURN> when ready');
      readln;
      acertorigin;
      bcertorigin;
    END;
    4 : BEGIN
      write(at(0,3),
        'Phytosanitary Certificate sheet, press <RETURN> when
          ready ');
      readln;
      aphytocert;
      bphytocert
    END;
    5 : BEGIN
      write(at(0,3),
        'Fumigation Certificate sheet, press <RETURN> when
          ready ');
      readln;
      fumigation
    END
  END;
  write(at(0,3),' ':78);
  write(at(0,3),'Repeat?(y/n)'); read(ch);
  IF ch IN ['Y','y'] THEN casecnt := casecnt - 1

```

```
UNTIL casecnt = 5;CLOSE(out)
END; (* shippaperwork *)
```

```

SEGMENT PROCEDURE fspshipment;
VAR choice, loc, sfloc, i, j, m, old, mm : integer;
    done : boolean;
    inp : string[10];
    shpamtintons, rate, rate1, rate2, cutwgt : real;
    ch : char;

PROCEDURE truckconvert(line:characters; VAR tr : trucktype);
CONST blank = ' ';
VAR temp : trucktype;
    i, j, max, start : integer;
    gross, tare : real;
    item : characters;
BEGIN
    line := CONCAT(line, '$');
    WITH temp DO
        BEGIN
            mthday := blank; wgtticketno := blank; net := 0; bales := 0;
        END;
    i := 1; j := 1; max := LENGTH(line);
    REPEAT
        WHILE(line[i] = blank) AND (i < max) DO
            i := i + 1;
        start := i;
        WHILE(line[i] <> blank) AND (i < max) DO
            i := i + 1;
        item := COPY(line, start, i-start);
        CASE j OF
            1 : temp.mthday := item;
            2 : temp.wgtticket := item;
            3 : temp.bales := conint(item);
            4 : gross := conreal(item);
            5 : BEGIN
                    tare := conreal(item);
                    temp.net := gross - tare
                END
        END; (* case *)
        j := j + 1
    UNTIL (j > 5) OR (i = max);
    tr := temp
END;

PROCEDURE gettruckrate;
VAR j : integer; inp : characters;

```

```

BEGIN
  j := 0;
  REPEAT
    j := j + 1;
    CASE j OF
      1 : BEGIN
        write(chr(12),at(0,2),'Cutting point weight:');
        readln(inp);
        IF inp <> ' '
          THEN cutwgt := conreal(inp)
          ELSE j := j - 1
        END;
      2 : BEGIN
        write(at(0,4),'Rate below cut point:');
        readln(inp);
        IF inp <> ' '
          THEN rate1 := conreal(inp)
          ELSE j := j - 1
        END;
      3 : BEGIN
        write(at(0,6),'Rate above cut point:');
        readln(inp);
        IF inp <> ' '
          THEN rate2 := conreal(inp)
          ELSE j := j - 1
        END;
    END; { case }
  END; { gettruckrate }

PROCEDURE truckcostcomp;
VAR truckmenu : menutype;
    ch : char;
    choice : integer;
BEGIN
  truckmenu[0] := 'Truck rate computation';
  truckmenu[1] := 'By shorttons';
  truckmenu[2] := 'By bales';
  fspshiprec.totaltruckcost := 0;
  REPEAT
    prompt(truckmenu,2,choice);
    write(at(0,10),'Compute ',truckmenu[choice],' is it correct?
                                                (y/n)');
    read(ch)
  UNTIL ch IN ['Y','y'];
  WITH fspshiprec DO

```

```

BEGIN
CASE choice OF
1 : BEGIN
    { get the rates }
    REPEAT
        gettruckrate;
        write(at(0,22),'Input OK?(y/n)');
        read(ch); write(at(0,22),' ':30)
    UNTIL ch IN ['Y','y'];
    { compute }
    FOR i := 1 TO noftruck DO
    BEGIN
        IF truck[i].net/2000.0 < cutwgt
            THEN truck[i].cost := (truck[i].net/2000.0) * rate1
            ELSE truck[i].cost := (truck[i].net/2000.0) * rate2;
        totaltruckcost := totaltruckcost + truck[i].cost
    END
    END;
2 : BEGIN
    { get the rates }
    REPEAT
        gettruckrate;
        write(at(0,22),'Input OK?(y/n)');
        read(ch); write(at(0,22),' ':30)
    UNTIL ch IN ['Y','y'];
    { compute }
    FOR i := 1 TO noftruck DO
    BEGIN
        IF truck[i].bales < cutwgt
            THEN truck[i].cost := rate1
            ELSE truck[i].cost := (truck[i].bales/cutwgt) * rate2;
        totaltruckcost := totaltruckcost + truck[i].cost
    END
    END
    END; { case }
    END; { with }
END;

PROCEDURE fspcompute;
BEGIN
WITH fspcontract DO
BEGIN
    i := 1; (* find out which months *)
    write(chr(12),at(18,0),'month      @price');
    REPEAT

```

```

    write(at(20,i),timeofship[i].month:2,'    $',
          timeofship[i].unitprice:7:2);
    i := i + 1;
UNTIL (i > 6) OR (timeofship[i].month = 0);

REPEAT
  write(at(20,10),'Month = ');
  readln(inp);
  IF inp <> ''
    THEN BEGIN
      mm := conint(inp);
      i := 0;
      REPEAT
        i := i + 1; m := i;
      UNTIL (timeofship[i].month = mm) OR (i = 7);
      IF i = 7
        THEN BEGIN
          write(at(20,12),'No such month listed,
                    press <RETURN>');
          readln
          END
        END
      UNTIL (timeofship[i].month = mm) AND (inp <> '');

  shpamtintons := fspshiprec.totalnet / 2000;
  timeofship[m].bal := timeofship[m].bal - shpamtintons;
  balofship := balofship - shpamtintons;
  fspshiprec.payment := shpamtintons * timeofship[m].unitprice;

truckcostcomp; { compute the truck cost }

{ now save into the fspshipfile first }
fspshipfile^ := fspshiprec;
SEEK(fspshipfile,sfloc);
PUT(fspshipfile); CLOSE(fspshipfile);

{ output some information }
write(chr(12),at(0,2),'Total of ',shpamtintons:8:3,' shorttons');
write(at(0,4),'Payment is $',fspshiprec.payment:10:2);
write(at(0,6),'Press <RETURN>'); readln;

IF (timeofship[m].bal < 0) AND (timeofship[m+1].month <> 0)
  THEN timeofship[m+1].bal := timeofship[m+1].bal +
    timeofship[m].bal;

```

```

IF nofshipment <> 0
  THEN BEGIN                                (* more than one shipments so *)
    i := 0; j := shipmentinfo; (* link up the shipment records *)
    RESET(fspshipfile, '#5:fspshipfile');
    REPEAT
      SEEK(fspshipfile, j); old := j;
      GET(fspshipfile);
      j := fspshipfile^.link; i := i + 1
    UNTIL i := nofshipment;
    fspshipfile^.link := sfloc;
    SEEK(fspshipfile, old); PUT(fspshipfile);
    CLOSE(fspshipfile); (* put back into the original place *)
    END
  ELSE shipmentinfo := sfloc; (* first shipment *)
  nofshipment := nofshipment + 1;
END; (* with *)
END; (* fspcompute *)

```

```

PROCEDURE getpurshipinfo;
VAR k, lineno, xaxis : integer;
    ch : char;
    finish, goon : boolean;
    temp : characters;
    tempdate : datatype;

```

```

FUNCTION psproceed : boolean;
BEGIN
  IF EOF THEN BEGIN RESET(INPUT); EXIT(fspshipment) END
  ELSE IF temp = ' '
    THEN BEGIN
      psproceed := false;
      lineno := lineno - 1
    END
  ELSE psproceed := true
END;

```

```

PROCEDURE nextpurshipinput;
BEGIN
  WITH fspshiprec DO
  BEGIN
    write(at(0,2),
    ' Date      Wgt ticket #   Bales   Gross   Tare   Net ');
    write(at(0, lineno+2), lineno:2, '. ');
    readln(temp);
    finish := (temp = 'F') OR (temp = 'f'0);

```

```

IF (psproceed) AND (NOT finish)
  THEN BEGIN
    truckconvert(temp, truck[lineno]);
    write(at(62, lineno+2), truck[lineno].net:8:1);
    totalbales := totalbales + truck[lineno].bales;
    totalnet := totalnet + truck[lineno].net;
    IF lineno > noftruck THEN noftruck := lineno
      {nessary not to reset noftruck when called from
        purshipmodify}
    END;
  IF finish
    THEN write(at(0, lineno+2), ' ':70)
  END { with }
END; (* nextpurshipinput *)

PROCEDURE purshipmodify;
BEGIN
  REPEAT
    REPEAT
      goon := true;
      write(at(55,22), 'Which line to change:0;
      readln(lineno); write(at(55,22), ' ':24);
      IF (lineno < 1) OR (lineno > fspshiprec.noftruck)
        THEN BEGIN
          write(at(50,22), chr(7), 'No such line, press
                                <RETURN>');
          readln; write(at(50,22), ' ':25); goon := false
        END
    UNTIL goon;
    (* now erase the line to be changed *)
    WITH fspshiprec DO
      BEGIN
        write(at(4, lineno+2), ' ':70);
        totalbales := totalbales - truck[lineno].bales;
        totalnet := totalnet - truck[lineno].net
      END;
    (* read the modifying line *)
    nextpurshipinput;
    write(at(55,22), 'OK now?(y/n)'); read(ch)
  UNTIL ch IN ['Y', 'y']
END; (* purshipmodify *)

BEGIN (* getpurshipinfo *)
  lineno := 1; fspshiprec.totalbales := 0;
  fspshiprec.totalnet := 0; fspshiprec.noftruck := 0;

```

```

write(chr(12)); { clear screen }
REPEAT
    nextpurshipinput;
    lineno := lineno + 1
UNTIL (lineno > 20) OR (finish);
fspshiprec.status := occupied;
write(at(55,22),'Input OK?(y/n)');read(ch);
IF ch IN ['N','n'] THEN purshipmodify
END;

BEGIN (* fspshipment *)
    RESET(fsphashfile,'#5:fsphashfile');
    REPEAT (* till the correct contract no given *)
        done := true;
        REPEAT
            write(chr(12),at(x,3),'Contract number:');
            readln(contractno)
        UNTIL (contractno <> ' ') OR EOF;
        IF EOF THEN BEGIN RESET(INPUT);CLOSE(fsphashfile);
        EXIT(fspshipment) END;
        i := -1;
        REPEAT
            i := 1 = 1;
            SEEK(fsphashfile,i);
            GET(fsphashfile)
        UNTIL (EOF(fsphashfile)) OR (fsphashfile^.number =
            contractno);

        IF EOF(fsphashfile)
        THEN REPEAT
            prompt(error2,3,choice);
            CASE choice OF
                1 : BEGIN CLOSE(fsphashfile);EXIT(fspshipment) END;
                2 : BEGIN CLOSE(fsphashfile);fspinquery END;
                3 : done := false
            END;
            UNTIL choce = 3
        ELSE BEGIN (* ok find the contract info from fspfile *)
            loc := fsphashfile^.link; CLOSE(fsphashfile);
            RESET(fspfile,'#5:fspfile');
            SEEK(fspfile,loc);
            GET(fspfile); fspcontract := fspfile^;
            CLOSE(fspfile)
            END;
        UNTIL done;

```

```

getpurshipinfo;
(* put this info into the fspshipfile *)
RESET(fspshipfile, '#5: fspshipfile'); sfloc := -1;
REPEAT
    sfloc := sfloc + 1;
    SEEK(fspshipfile, sfloc);
    GET(fspshipfile)
UNTIL (EOF(fspshipfile)) OR (fspshipfile^.status = empty);
write(at(0,22), 'sfloc = ', sfloc);

```

```

IF EOF(fspshipfile)
    THEN BEGIN
        CLOSE(fspshipfile);
        write(chr(12), chr(7), at(x,2),
            'DOOMESDAY no more space in the file');
        write(at(x,3), 'Please call the system designer');
        write(at(x,4), 'Meantime press <RETURN>
            and do other work');
        readln; EXIT(fspshipment)
    END;

```

```

(* make the necessary computation and save it *)
fspcompute;

```

```

(* now put it into the fspfile *)
RESET(fspfile, '#5: fspfile');
fspfile^ := fspcontract;
SEEK(fspfile, loc); PUT(fspfile);
CLOSE(fspfile);
END; (* fspshipment *)

```

```

SEGMENT PROCEDURE listpurcontr; (*purchase contract info *)
VAR k : integer;
BEGIN
    WITH fspcontract DO
        BEGIN
            write(chr(12), at(15,0), 'Feed Stuff Purchase Contract
                Information ');
            write(at(x,1), fspnewconmenu[1], ' ', number);
            write(at(x,2), fspnewconmenu[2], ' ', contrdate.month, '/',
                contrdate.day, '/', contrdate.year);
            write(at(x,3), fspnewconmenu[3], ' ', farmer.name);
            write(at(x,4), fspnewconmenu[4], ' ', farmer.addr);
            write(at(x,5), fspnewconmenu[5], ' ', commodity);

```

```

k := 1;
write(at(x,6),
'Time of shipment : Months Quantity Balance Unitprice');
REPEAT
  write(at(29+x,6+x),timeofship[k].month:2,timeofship[k].wgt
    :10,timeofship[k].bal:12:2,' $',timeofship[k].unitprice:8:2);
  k := k + 1
UNTIL (timeofship[k].month = 0) OR (k > 6);
write(at(x,6+k),'total shipment :',totalship:8:2);
write(at(x,7+k),'balance of shipment:',balofship:8:2);
write(at(x,8+k),'# of shipment made:',nofshipment)
END;
END; (* listpurcontr *)

```

```

SEGMENT PROCEDURE prlistpurcontr; (* contract infor to printer *)
VAR k : integer;
BEGIN
  WITH fspcontract DO
  BEGIN
    skip(4);
    writeln(out,' :20,'Feed Stuff Purchase Contract Information');
    skip(3);
    writeln(out,' :10,fspnewconmenu[1],' ',number);writeln(out);
    writeln(out,' :10,fspnewconmenu[2],' ',contrdate.month,'/',
      contrdate.day,'/',contrdate.year); writeln(out);
    writeln(out,' :10,fspnewconmenu[3],' ',farmer.name);
    writeln(out);
    writeln(out,' :10,fspnewconmenu[4],' ',farmer.addr);
    writeln(out);
    writeln(out,' :10,fspnewconmenu[5],' ',commodity);
    writeln(out);
    skip(2); k := 1;
    writeln(out,' :10,
'Time of shipment : Months Quantity Balance Unitprice');
    REPEAT
      writeln(out,' :39,timeofship[k].month:2,timeofship[k].wgt:10,
        timeofship[k].bal:12:2,' $',timeofship[k].unitprice:8:2);
      k := k + 1
    UNTIL (timeofship[k].month = 0) OR (k > 6); skip(2);
    writeln(out,' :10,'total shipment :',totalship:8:2);
    writeln(out);
    writeln(out,' :10,'balance of shipment :',balofship:8:2);
    writeln(out);
    writeln(out,' :10,'# of shipment made :',nofshipment)
  END;
END;

```

```
END; (* prlistpurcontr *)
```

```
SEGMENT PROCEDURE purcontrinfo;  
VAR i,j,k,num,entries,loc : integer;  
    commoname : string[25];  
    quit : boolean;  
    commodlist : intype;  
    ch : char;  
    twodeci,str1,str2,str3 : characters;
```

```
PROCEDURE listpurship;
```

```
BEGIN
```

```
    WITH fspshiprec DO
```

```
    BEGIN
```

```
        write(chr(12),at(0,0),'Shipment No. ',num:2);  
        write(at(5,1),'Date',at(18,1),'Wgt tkt#',at(33,1),'Bales',at(41,1),  
            'Net Wgt',at(95,1),'Cost');
```

```
    FOR k := 1 TO noftruck DO
```

```
        WITH truck[k] DO
```

```
        BEGIN
```

```
            prealtostr(net,str3);addcomma(str3);  
            write(at(5,2+k),mthday,at(18,2+k),wgtticketno,  
                at(31,2+k),bales:5,at(40,2+k),str3:7,cost:9:2)
```

```
        END;
```

```
    STR(totalbales,str1);addcomma(str1);  
    write(at(0,4+noftruck),'Total bales =',str1);  
    prealtostr(totalnet,str2);addcomma(str2);  
    write(0,5+noftruck),'Total net =',str2);  
    dollarcent(payment,twodeci);addcomma(twodeci);  
    write(at(0,6+noftruck),'Payment =$',twodeci);  
    dollarcent(totaltruckcost,str3);addcomma(str3);  
    write(at(0,8+noftruck),'truck cost =$',str30;
```

```
END
```

```
END;
```

```
PROCEDURE printpurship;
```

```
BEGIN
```

```
    WITH fspshiprec DO
```

```
    BEGIN
```

```
        write(at(0,22),' ':70);  
        write(at(0,22),'Need a printout?(y/n)');  
        read(ch);IF EOF THEN BEGIN RESET(INPUT);EXIT(printpurship)
```

```
    END;
```

```
    IF ch IN ['Y','y']
```

```
        THEN BEGIN
```

```

write(at(0,22),'Turn on the TEC and press <RETURN>');
readln;
REWRITE(out,'PRINTER:'); skip(4);
writeln(out,'  Shipment No.',num:2);writeln(out);
writeln(out,' ':4,'Date',' ':14,'Wgt ticket#',' ':5,'Bales',
          ' ':5,'Net Wgt',' ':5,'Cost'); skip(2);
FOR k := 1 TO noftruck DO
  WITH truck[k] DO
    BEGIN
      prealtostr(net,str3);addcomma(str3);
      writeln(out,' ':4,mthday,' ':12,wgtticketno:12,
                ' ':5,bales:5,' ':5,str3:7,' ':5,cost:7:2);
      writeln(out)
    END;
  STR(totalbales,str1);addcomma(str1);
  prealtostr(totalnet,str2);addcomma(str2);
  dollarcent(totaltruckcost,str3);addcomma(str3);
  dollarcent(payment,twodeci);addcomma(twodeci);
  writeln(out);
  writeln(out,' ':5,'Total bales =',str1);writeln(out);
  writeln(out,' ':5,'Total net   =',str2);writeln(out);
  writeln(out,' ':5,'Payment   =$',twodeci:10);writeln(out);
  writeln(out,' ':5,'Truck cost =$',str3:10);
COLSE(out)
END
END;  (* with *)
END;

```

```

PROCEDURE subpurcontr; { for shipment info }
BEGIN
  IF fspcontract.nofshipment > 0 THEN
    BEGIN
      write(at,(0,22),
        'Like to see all shipments in sequence?(y/n/<ctrl-c> to quit)');
      read(ch);IF EOF THEN BEGIN RESET(INPUT);EXIT(subpurcontr)
        END;
      IF ch IN ['Y','y']
        THEN BEGIN
          i := 0;j := fspcontract.shipmentinfo;
          RESET(fspshipfile,'#5: fspshipfile');
          REPEAT
            SEEK(fspshipfile,j);
            GET(fspshipfile);
            j := fspshipfile^.link;
            fspshiprec := fspshipfile^;

```

```

num := i + 1; { used in printing proc }
listpurship; printpurship; i := i + 1;
IF i < fspcontract.nofshipment
  THEN BEGIN
    write(at(0,22),'Want to see next shipment?
           (y/n) ');
    read(ch);write(at(0,22),' ':50);
    IF EOF THEN BEGIN CLOSE(fspshipfile);
                      RESET(INPUT);
                      EXIT(subpurcontr) END
    END
  UNTIL (i = fspcontract.nofshipment) OR (ch IN ['N','n']);
  CLOSE(fspshipfile);
  END
ELSE BEGIN
  REPEAT
    write(at(0,220),' ':70); (* erase previous line *)
    REPEAT
      write(at(0,22),'Which shipment(<ctrl-c> to quit):');
      read(num); IF EOF THEN BEGIN RESET(INPUT);
                                EXIT(subpurcontr)
                                END
    UNTIL (num>=1) AND
           (num<=fspcontract.nofshipment);
    (* locat the shipment info *)
    i := 0; j := fspcontract.shipmentinfo;
    RESET(fspshipfile,'#5:fspshipfile');
    REPEAT
      SEEK(fspshipfile,j);
      GET(fspshipfile);
      j := fspshipfile^.link; i := i + 1
    UNTIL i:= num;
    fspshipprec := fspshipfile^; CLOSE(fspshipfile);
    listpurship; printpurship;
    write(at(0,22),'Like to see another shipment?(y/n)');
    read(ch);
    IF EOF THEN BEGIN RESET(INPUT);EXIT(subpurcontr)
    END
  UNTIL ch IN ['y','Y']
  END (* else *)
END
END;

BEGIN (* purcontrinfo *)
  REPEAT

```

```

    write(chr(12),at(x,3),'Contract number:');
    readln(contractno);
UNTIL (contractno <> ' ') OR EOF;(* not empty or terminate *)
IF EOF THEN BEGIN RESET(INPUT);EXIT(purcontrinfo) END;
RESET(fsphashfile,'#5:fsphashfile'); i := - 1;
REPEAT
    i := i + 1;
    SEEK(fsphashfile,i);
    GET(fsphashfile)
UNTIL (EOF(fsphashfile) OR (fsphashfile^.number = contractno);

IF EOF(fsphashfile) (* not found *)
THEN BEGIN
    CLOSE(fsphashfile);
    write(at(x,5),chr(70,'No such contract in the file');
    write(at(x,7),'Press <RETURN>'); readln;
    END
ELSE BEGIN
    CLOSE(fsphashfile);
    loc := fsphashfile^.link;
    RESET(fspfile,'#5:fspfile');
    SEEK(fspfile,loc);GET(fspfile);CLOSE(fspfile);
    fspcontract := fspfile^.
    listpurcontr;
    { prntout to the printer if desired }
    write(at(0,22),'Need a printout?(y/n));read(ch);
    IF EOF THEN BEGIN RESET(INPUT);EXIT(purcontrinfo) END;
    IF ch IN ['Y','y']
        THEN BEGIN
            write(at(0,22),'Turn on the TEC and press
                                <RETURN>');
            readln;REWRITE(out,'PRINTER:');
            prlistpurcontr; CLOSE(out)
            END;
        subpurcontr { to list/print shipment information }
    END;
END;

SEGMENT PROCEDURE fspresiduecheck;
VAR j,opencnt : integer;
BEGIN
    write(chr(12),at(x,2),'Available Spaces'); opencnt := 0;
    write(at(x,4),'Contract file: ');
    j := 0; RESET(fspfile,'#5:fspfiel');
    SEEK(fspfile,j); GET(fspfile);

```

```

WHILE NOT EOF(fspfile) DO
BEGIN
  IF fspfile^.status = empty
  THEN BEGIN
    opencnt := opencnt + 1;
    write(at(x+19,4),opencnt:3)
    END;
    j := j + 1;
    SEEK(fspfile,j);
    GET(fspfile)
  END;
  CLOSE(fspfile);
  opencnt := 0; j := 0;
  write(at(x,5),'Shipment file: ');
  RESET(fspshipfile,'#5:fspshipfile');
  SEEK(fspshipfile,j); GET(fspshipfile);
  WHILE NOT EOF(fspshipfile) DO
  BEGIN
    IF fspshipfile^.status = empty
    THEN BEGIN
      opencnt := opencnt + 1;
      write(at(x+19,5),opencnt:3)
      END;
      j := j + 1;
      SEEK(fspshipfile,j);
      GET(fspshipfile)
    END;
    CLOSE(fspshipfile);
    write(at(x,7),'Press <RETURN>'); readln
  END; { fspresiduecheck }

```

```

SEGMENT PROCEDURE fspinquiry;
VAR i,j,k,num,entries,loc : integer;
    commoname : string[25];
    quit : boolean;
    commodlist : intype;
    ch : char;
    twodeci,str1,str2,str3 : characters;
BEGIN (* fspinquiry *)
  quit := false;
  REPEAT
    prompt(fspquerymenu,5,choice);
    CASE choice OF
      1 : BEGIN
        RESET(fsphashfile,'#5:fsphashfile');

```

```

write(chr(12),at(15,0),'List of all commodities');
entries := 1; commodlist[1] := ' '; j := -1;
REPEAT
  REPEAT
    j := j + 1;
    SEEK(fsphashfile,j);
    GET(fsphashfile)
  UNTIL (EOF(fsphashfile)) OR (fsphashfile^.status <>
                                empty);
  IF NOT EOF(fsphashfile)
  THEN BEGIN
    i := 1;
    (* check if this record's name is already in
       commodity array, if not put it *)
    WHILE (i < entries) AND
      (fsphashfile^.commodity <> commodlist[i]) DO
      i := i + 1;
    IF i := entries
    (* not in customers array so put it in *)
    THEN BEGIN
      commodlist[i] :=
        fsphashfile^.commodity;
      entries := entries + 1
    END;
  END
UNTIL EOF(fsphashfile);
CLOSE(fsphashfile); i := 1;
WHILE i <= entries - 1 DO
BEGIN
  IF validate(commodlist[i])
  THEN write(at(x,i),commodlist[i]);
  i := i + 1
END;
write(at(55,22),'Press <RETURN>'); readln
END;
2 : BEGIN
  REPEAT
    write(chr(12),at(x,3),'Commodity name:');
    readln(commoname)
  UNTIL (commoname <> ' ') OR EOF;
  (* not empty or terminate *)
  IF EOF THEN BEGIN RESET(INPUT);EXIT(fspinquiry) END;
  RESET(fsphashfile,'#5:fsphashfile');
  write(chr(12),at(15,0),'Contracts of',commoname);
  j := 2;

```

```

FOR i := 0 TO max DO
BEGIN
    SEEK(fsphashfile,i);
    GET(fsphashfile);
    IF (fsphashfile^.status = occupied)
        AND (fsphashfile^.commodity = commoname)
    THEN BEGIN
        write(at(x,j),fsphashfile^.number,' ',
            fsphashfile^.name);

        j := j + 1
    END;
    IF j = 22 (* full screen *)
    THEN BEGIN
        write(at(55,22),'Press <RETURN>');
        readln; write(chr(12); j := 2
        END;
    END;
    write(at(55,22),'Press <RETURN>'); readln;
    CLOSE(fsphashfile)
END;
3 : purcontrinfo;
4 : fspresiduecheck;
5 : quit := true;
END; (* case *)
UNTIL quit
END;

SEGMENT PROCEDURE fspnew;
(* to add new feed stuff contract and setup the data structure
    accordingly *)
VAR lineno,j,k,loc,choice,addr,i : integer;
    goon,finish,done,located : boolean;
    temp : characters;
    ch : char;
    tempdate : datatype;

PROCEDURE toconvert(line:characters;VAR tos:tostype);
CONST blank = ' ';
VAR temp : tostype;
    i,j,max,start : integer;
    item : characters;
BEGIN
    line := CONCAT(line,'$');
    max := LENGTH(line);
    WITH temp DO

```

```

BEGIN
    month := 0; wgt := 0; bal := 0; unitprice := 0
END;
i := 1; j := 1;
REPEAT
    WHILE(line[i] = blank) AND (i < max) DO
        i := i + 1;
    item := COPY(line,start,i-start);
    CASE j OF
        1 : temp.month := conint(item);
        2 : BEGIN
            temp.wgt := conint(item);
            temp.bal := temp.wgt
        END;
        3 : temp.unitprice := conreal(item)
    END; (* case *)
    j := j + 1
UNTIL (j > 3) OR (i = max);
tos := temp
END; (* toconvert *)

PROCEDURE getfspinfo;
(* get all the information for the new contract *)

FUNCTION pproceed : boolean;
BEGIN
    IF EOF THEN BEGIN RESET(INPUT);EXIT(fspnew) END
    ELSE IF (lineno <> 6) AND (temp = ' ')
        THEN BEGIN
            pproceed := false;
            lineno := lineno - 1
        END
    ELSE IF (lineno = 6) AND (temp = ' ')
        THEN BEGIN
            pproceed := false;
            k := k - 1
        END
    ELSE pproceed := true
END;

PROCEDURE nextpurinput;
BEGIN
    WITH fspcontract DO
        CASE lineno OF
            1,3,4,5:

```

```

BEGIN
write(at(x,lineno),lineno:2,' ',fspnewconmenu[lineno]);
readln(temp);
IF pproceed
    THEN CASE lineno OF
        1 : number := temp;
        3 : farmer.name := temp;
        4 : farmer.addr := temp;
        5 : commodity := temp;
        END; (* case *)
    END;
2 : BEGIN
write(at(x,2),' 2. ',fspnewconmenu[2]);
readln(temp);
IF pproceed
    THEN BEGIN
        dateconvert(temp,contrdate);
        IF datecheck(contrdate) <> ok
            THEN BEGIN
                write(at(38,2),'Error in input, press
                    <RETURN>');
                readln;write(at(38,2),' ':30);
                lineno := lineno - 1
            END
        END
    END;
END;
6 : BEGIN
    WITH fspcontract DO
    BEGIN
        totalship := 0;
        write(at(x,9),' 6. ',fspnewconmenu[6]);
        finish := false; k := 0;
        REPEAT
            k := k + 1;
            GOTOXY(x+33,9+k); readln(temp);
            finish := (temp = 'F') OR (temp = 'f');
            IF (pproceed) AND (NOT finish)
                THEN BEGIN
                    tosconvert(temp,timeofship[k]);
                    IF (timeofship[k].month < 1) OR
                        (timeofship[k].month > 12)
                        THEN BEGIN
                            write(at(x+30,9+k),
                                'Error in input,press <RETURN>');
                            readln;

```

```

                                write(at(x+30,9+k),' ':30); k := k - 1
                                END
                                ELSE totalship := totalship +
                                    timeofship[k].wgt
                                END;
                                UNTIL (k = 6) OR (finish);
                                balofship := totalship;
                                IF k < 6 THEN BEGIN
                                    write(at(x+30,9+k),' ':30);
                                    timeofship[k].month := 0
                                    (* o is endofdata marker *)
                                    END
                                END
                                END;
                                END;
                                END; (* case *)
                                END; (* nextpurinput *)

PROCEDURE fspmodify;
(* to modify the fspcontract input information *)
BEGIN
    REPEAT
        REPEAT
            goon := true;
            write(at(55,22),'Which line to change:');
            readln(lineno); write(at(55,22),' ':24);
            IF (lineno < 1) OR (lineno > 6)
                THEN BEGIN
                    write(at(55,22),chr(7),'No such line!Press <RET>');
                    readln;write(at(55,22),' ':25);goon := false
                END
            UNTIL goon;
            (* now erase the line to be changed *)
            IF lineno <= 5
                THEN BEGIN
                    write(at(38,lineno),' ':40);
                    GOTOXY(38,lineno)
                END
            ELSE FOR j := 1 TO k DO
                    write(at(43,9+j),' ':20);
                    (* no GOTOXY here since it is in case 6: *)
                nextpurinput;
                write(at(55,22),'OK now?(y/n)');
                read(ch);
            UNTIL (ch = 'Y') OR (ch = 'y')
        END; (* fspmodify *)

```

BEGIN

```
  write(chr(12),at(15,0),fspnewconmenu[0]);
  lineno := 1;
  REPEAT
    nextpurinput;
    lineno := lineno + 1
  UNTIL lineno > 6;
  fspcontract.nofshipment := 0; fspcontract.status := occupied;
  write(at(55,22),'Input OK?(y/n)');
  read(ch); IF (ch = 'N') OR (ch = 'n') THEN fspmodify
END; (* getfspinfo *)
```

BEGIN (* fspnew *)

```
  getfspinfo;(* input all pertinent new purchase contract info *)
  (* go through the file and make sure that the given contract#
    is not already in the file *)
```

```
  RESET(fsphashfile,'#5:fsphashfile');
```

```
  REPEAT
```

```
    done := true;
```

```
    i := -1;
```

```
    REPEAT
```

```
      i := i + 1;
```

```
      SEEK(fsphashfile,1);
```

```
      GET(fsphashfile)
```

```
    UNTIL (EOF(fsphashfile)) OR
```

```
      (fsphashfile^.number = fspcontract.number);
```

```
  IF fsphashfile^.number = fspcontract.number
```

```
    THEN REPEAT (* error! same contract already in table *)
```

```
      prompt(error1,3,choice);
```

```
      CASE choice OF
```

```
        1 : BEGIN CLOSE(fsphashfile);EXIT(fspnew) END;
```

```
        2 : BEGIN CLOSE(fsphashfile);fspinquery END;
```

```
        3 : BEGIN
```

```
          REPEAT
```

```
            write(chr(12),at(x,3),'Contract number
              (<ctrl-c> to quit):');
```

```
            readln(fspcontract.number);
```

```
          UNTIL (fspcontract.number <> ' ') OR EOF;
```

```
          done := false;
```

```
          IF EOF THEN BEGIN
```

```
            CLOSE(fsphashfile);
```

```
            RESET(INPUT);
```

```
            EXIT(fspnew)
```

```

                                END
                                END;
                                END; (* case *)
                                UNTIL choice = 3
(* no error so put into the fspfile and fsphashfile *)
ELSE BEGIN
    (* place the new purchase contract info into the fspfile;
       place at the first open slot *)
    RESET(fspfile, '#5:fspfile');
    (* put the contract info into the first open slot *)
    loc := -1;
    REPEAT
        loc := loc + 1;
        SEEK(fspfile, loc);
        GET(fspfile);
    UNTIL (fspfile^.status = empty) OR (EOF(fspfile));
    IF EOF(fspfile)
    THEN BEGIN
        write(chr(12), at(x, 3), 'DOOMESDAY! No more space
                                to add new contract');
        write(at(x, 4), 'Must use new diskette.
                                Press <RETURN>');
        readln; CLOSE(fspfile); EXIT(fspnew)
    END;
    fspfile^:= fspcontract;
    SEEK(fspfile, loc);
    PUT(fspfile); CLOSE(fspfile);
    write(at(0, 22), 'loc = ', loc);

    (* find open slot in fsphashfile *)
    RESET(fsphashfile);
    i := -1;
    REPEAT
        i := i + 1;
        SEEK(fsphashfile, i);
        GET(fsphashfile)
    UNTIL fsphashfile^.status = empty;
    (* put in the information *)
    WITH fsphashfile^ DO
    BEGIN
        status := occupied;
        number := fspcontract.number;
        name := fspcontract.farmer.name;
        link := loc;
        commodity := fspcontract.commodity

```

```
END;  
SEEK(fsphashfile,i);  
PUT(fsphashfile);CLOSE(fsphashfile)  
END  
UNTIL done  
END; (* fspnew *)
```

APPENDIX B

THE dBASE III PLUS PROGRAMS

```
*****          program HANAOKA
*****          Original written in Apple Pascal
*****          Rewritten in dBASE III PLUS
*****          Author : To Chang
*****          Date : June 1987
*****          Instructor : Prof. Thomas C. Wu
```

```
clear all
set talk off
set bell off
set dele on
set exact on
```

```
store .T. to badentry
```

```
do while badentry
  clear
  @ 10,10 say '          Password : (or hit <CR> to exit...)'
  @ 10,50
```

```
set escape off
set exact on
set console off
accept to mpass
set console on
```

```
if mpass = ' '
  set escape on
  set exact off
  clear
  return
endif
```

```
if mpass # 'HANAOKA'
  @ 20,20 say 'Incorrect password ... (hit <CR> to retry)'
  ? chr(7)
  wait ' '
  loop
endif
store .F. to badentry
```

enddo

MAIN MENU

clear

store ' ' to mchoice

@ 02,15 say '++++++'

@ 03,15 say '+'

@ 04,15 say '+' ABC COMPANY '+'

@ 05,15 say '+'

@ 06,15 say '++++++'

@ 08,15 say ' 1. Sales'

@ 10,15 say ' 2. Purchase'

@ 12,15 say ' 3. Quit'

@ 18,15 say ' Choose one function -->(1,2,3)';

get mchoice pict 'x'

read

store .T. to mcontinue

do while mcontinue

do case

case mchoice = '1'

do sale

case mchoice = '2'

do purchase

otherwise

store .F. to mcontinue

clear

endcase

enddo

close databases

quit

* <End of HANAOKA>

```

*Program SALE.PRG ----- called from HANAOKA
do while .T.
clear
store ' ' to mchoice
@ 02,15 say '+++++'
@ 03,15 say '+'
@ 04,15 say '+'          ABC   Sale Information
@ 05,15 say '+'
@ 06,15 say '+++++'
@ 08,15 say '          1. New Sale Contract Entry'
@ 10,15 say '          2. New Sale Shipment Entry'
@ 12,15 say '          3. Sale Information Inquiry'
@ 14,15 say '          4. Quit'
@ 18,15 say '          Choose one function -->(1,2,3,4);
                                get mchoice pict 'x'

read
do case
    case mchoice = '1'
        do newsale
    case mchoice = '2'
        do s_shipinfo
    case mchoice = '3'
        do s-inquiry
    otherwise
        clear
        return
endcase
enddo
*      <End of SALE>

```

* Program NEWSALE.PRG ----- called from SALE.PRG
clear

public mttotalship,mnofshipmnt,msnumber
do while .T.

store	space(12)	to	msnumber,mlcnumber
store	space(50)	to	mcommodity,mcustaddr
store	space(80)	to	mpricebase
store	space(30)	to	missuebank,mdrawbank
store	space(18)	to	mmitino
store	space(25)	to	mcname
store	space(13)	to	mcustphone
store	space(8)	to	mcontrdate,mlcexpdate,mlcshipdate
store	' '	to	manswer
store	0	to	mlcbal,mwgt,mbal,mlcamount
store	0	to	munitprice,mttotalship,mbalofship
store	0	to	mnofshipment
store	.T.	to	nogood,notok

do while notok

do while nogood

set confirm on

set format to contract

read

if msnumber = ' '

set format to

set confirm off

store .F. to nogood

return

endif

use b:s_contract index b:s_conindx

find &msnumber

if found()

? chr(7)

@ 04,50 say 'Duplicate key !'

@ 24,05 say 'Replace? Discard? Inquiry?(R,D,I)';

get manswer pict 'x'

read

if upper(manswer) = 'R'

store .F. to nogood

@ 24,05 say ' '

endif

if upper(manswer) = 'I'

use

close index

```

        do s_inquery
        return
    endif
    if upper(manswer) = 'D'
        clear all
    endif
else
    set format to
    store .F. to nogood
endif
enddo
store ' ' to manswer
? chr(7)
@ 24,05 say 'Input OK?(y/n)' get manswer pict 'x'
read
if upper(manswer) = 'Y'
    store .F. to notok
endif
enddo

```

do tmship

use
close index

use b:s_contract index b:s_conindx

append blank

replace snumber	with msnumber
replace cname	with mcname
replace contrdate	with mcontrdate
replace commodity	with mcommodity
replace pricebase	with mpricebase
replace lcnumber	with mlcnumber
replace lcexpdate	with mlcexpdate
replace lcshipdate	with mlcshipdate
replace lcbal	with mlcbal
replace lcamount	with mlcamount
replace totalship	with mttotalship
replace balofship	with mbalofship
replace issuebank	with missuebank
replace drawbank	with mdrawbank
replace mitino	with mmitino
replace nofshipmnt	with mnofshipmnt

```

use
close index

select 3
use b:customer index b:custindx
find &mcname
if .NOT. found()
    append blank
    replace cname           with mcname
    replace address        with mcustaddr
    replace phoneno        with mcustphone
endif
close databases
clear all
enddo
*      < End of NEWSALE.PRG >

```

```
* Program S_INQUERY.PRG ----- called from HANAOKA or
*                               or NEWSALE or S_SHIPINFO
```

```
do while .T.
clear
clear all
public mcontrno
store 0 to lctr
store ' ' to mchoice
store ' ' to mcontrno
store ' ' to mcompname
```

```
@ 02,15 say '+++++'
@ 03,15 say '+'
@ 04,15 say '+' SALE INFORMATION INQUIRY '+'
@ 05,15 say '+'
@ 06,15 say '+++++'
@ 08,15 say ' 1. List Customer'
@ 10,15 say ' 2. List All Contracts of One Customer'
@ 12,15 say ' 3. List One Contract Info.'
@ 14,15 say ' 4. Quit'
@ 22,15 say ' Choose one function --->(1,2,3,4);
              get mchoice pict 'x'
```

```
read
```

```
do case
  case mchoice = '1'
    clear
    @ 00,10 say '++++++ Customer List ++++++'
    use b:customer index b:custindx
    do while .not. eof()
      store lctr+1 to lctr
      if lctr >= 19
        store 0 to lctr
        wait 'More list on next page...' to memvar
        clear
      endif
      @ lctr,02 say 'Name: '
      @ lctr,08 say cname
      store lctr+1 to lctr
      @ lctr,02 say 'Address: '
      @ lctr,11 say address
      store lctr+1 to lctr
      @ lctr,02 say 'Phone number: '
      @ lctr,16 say phoneno
```

```

        store lctr+1 to lctr
        @ lctr,10 say '===== '
        skip
    enddo
wait
use
close index
case mchoice = '2'
clear
@ 02,1 say 'Company Name = ' get mcompname;
    pict 'xxxxxxxxxxxxx'
read
@ 04,15 say 'Contract with '
@ 04,30 say mcompname
use b:s_contract index b:s_conindx
disp all cname,commodity for cname = mcompname off
wait
use
close index
case mchoice = '3'
clear
@ 12,20 say 'Contract Number:' get mcontrno;
    pict 'xxxxxxxxxxxxx'
read
@ 12,37 say mcontrno
select 2
use b:s_contra index b:s_conindx

find &mcontrno
if found()
    use
    close index
    do salelist
    @ 23,25 say ''
    accept 'Need a printout?(y/n)' to manswer
    if upper(manswer) = 'Y'
        @ 23,00 say 'Turn on the printer,'
        wait
        set device to print
        do salelist
        set device to screen
    endif
    store ' ' to manswer
else
    @ 20,20 say 'No such contract in the file!!'

```

```
        ? chr(7)
        wait
    endif
    use
    close index
case mchoice = '4'
    return
endcase
enddo
*      <End of S_INQUERY>
```

* Program S_SHIPINFO ----- called from SALE.PRG

clear

store ' ' to manswer

do while .T.

store	space(15)	to minvoiceno
store	space(25)	to mname, morigin, mdest
store	space(8)	to metd, minvdate
store	space(12)	to msnumber, mcnumber
store	0	to mtotalbales, mtotalnet
store	0	to mnofcontner, mbales, mnet

set confirm on

set format to sshipment

read

if minvoiceno = ' '

set format to
set confirm off
use
close index
return

endif

@ 24,20 say 'Input OK?(y/n)' get manswer pict 'x'

read

if upper(manswer) = 'Y'

use b:contner index b:contnrndx
append blank
replace cnumber with msnumber;
invoiceno with minvoiceno

do scompute

replace bales with mbales;
net with mnet

store mtotalbales + mbales to mtotalbales
store mtotalnet + mnet to mtotalnet
store mnofcontner + 1 to mnofcontner

store .F. to mcontinue

@ 24,15 say 'More container?(y/n)-->' get manswer pict 'x'

read

if upper(manswer) = 'Y'

store .T. to mcontinue

```

        @ 24,15 say '
endif
do while mcontinue
    set format to contfmt
    read
    if mcnumber = '
        set format to
        set confirm off
        use
        close index
        store .F. to mcontinue
    else
        @ 24,15 say 'Input OK?(y/n)-->' get manswer;
            pict 'x'
        if upper(manswer) = 'Y'
            @ 24,15 say '
            appnd blank
            replace cnumber          with mcnumber;
                invoiceno          with minvoiceno

            do scompute

            replace bales            with mbales;
                net                with mnet

            store mtotalbales + mbales to mtotalbales
            store mtotalnet + mnet    to mtotalnet
            store mnofcontner + 1    to mnofcontner

            @ 24,15 say 'More containers?(y/n)-->' get;
                manswer pict 'x'
            read
            @ 24,15 say '
            if upper(manswer) = 'Y'
                store .T. to mcontinue
            else
                store .F. to mcontinue
            endif
        endif
    endif
enddo

use b:s_shipmnt index b:sshipdx

append blank

```

```

        replace invoiceno          with minvoiceno;
                name                with mname;
                origin              with morigin;
                dest                with mdest
replace etd                      with metd;
                invoicedat         with minvdate
replace totalbales              with mtotalbales;
                totalnet           with mtotalnet;
                nofcontrner        with mnofcontrner;
                snumber            with msnumbr

        use
        close index
endif
enddo
*           <End of S_SHIPINFO>
* Program SALELIST.PRG  -----   called from S_INQUERY.PRG

```

```

clear
set relation to cname into customer

```

```

@ 01,10 say '==== Feed Stuff Sale Contract Information ====='
@ 02,02 say 'Sale Number:'
@ 02,19 say snumber
@ 03,02 say 'Contract Date:'
@ 03,19 say contrdate
@ 04,02 say 'Customer Name:'
@ 04,19 say cname
@ 05,02 say 'Customer Address:'
@ 05,19 say address
@ 06,02 say 'Commodity:'
@ 06,19 say commodity
@ 07,02 say 'Price Base:'
@ 07,19 say pricebase
@ 08,02 say 'Time of Shipment           : Months   Quantity   '
@ 08,46 say ' Balance   Unitprice'

```

```

store 9 to mlnctr
set relation to snumber into tmofship

```

```

do while .not. eof(tmofship) .and. snumber = mcontrno
    @ mlnctr,32 say month
    @ mlnctr,39 say wgt
    @ mlnctr,47 say bal
    @ mlnctr,57 say unitprice

```

```

        store mlnctr + 1 to mlnctr
        skip
    enddo

    @ mlnctr+2,02 say 'Total Shipment:'
    @ mlnctr+2,22 say totalship
    @ mlnctr+3,02 say 'Balance of Shipment:'
    @ mlnctr+3,22 say balofship
    @ mlnctr+4,02 say 'L/C number:'
    @ mlnctr+4,22 say lcnumber
    @ mlnctr+5,02 say 'L/C expire date:'
    @ mlnctr+5,22 say lcexpdate
    @ mlnctr+6,02 say 'L/c shipment date:'
    @ mlnctr+6,22 say lcshipdate
    @ mlnctr+7,02 say 'L/C amount:'
    @ mlnctr+7,22 say lcamount
    @ mlnctr+8,02 say 'L/C balance:'
    @ mlnctr+8,22 say lcbal
    @ mlnctr+9,02 say 'Issue Bank:'
    @ mlnctr+9,22 say issuebank
    @ mlnctr+10,02 say 'Draw Bank:'
    @ mlnctr+10,22 say drawbank
    @ mlnctr+11,02 say 'Miti number:'
    @ mlnctr+11,22 say mitino
    @ mlnctr+12,02 say '# of shipment made:'
    @ mlnctr+12,22 say nofshipment

    return
    *      <End of SALELIST.PRG>

```

* Program SCOMPUTE.PRG ----- called from S_SHIPINFO.PRG

public mnet
public mrate

store 0 to mrate

select 2
use b:tmofship index b:tmshipdx
set relation to snumber into tmofship

do while .not. eof(tmofship) .and. snumber = msnumber
 if month = substr(invoicedate,4,5)
 @ 12,15 say 'Rate = '
 @ 12,20 say unitprice
 @ 20,10 say 'Compute the price with the above rate?(y/n)';
 get manswer pict 'x'
 read
 if upper(manswer) = 'N'
 clear
 @ 12,15 say 'Rate = ' get mrate pict '999999999999.99'
 else
 store unitprice to mrate
 endif

store totalnet/2000.0 to mshiintons
store bal-mshintons to bal

set relation to snumber into s_contra

store balofship-mshintons to balofship
store lcbal-mshintons*mrate to lcbal

endif

skip

enddo

use

close index

return

* <End of SCOMPUTE.PRG>

* Program SALEPRT.PRG ----- called from S_INQUERY.PRG

```
clear
store space(25) to mcname
select 2
use b:s_contract index b:s_conindx
find &mcontrno
@ 00,10 say '==== Feed Stuff Sale Contract Information ====='
@ 02,02 say 'Sale Number:'
@ 02,19 say snumber
@ 03,02 say 'Contract Date:'
@ 03,19 say contrdate
@ 04,02 say 'Customer Name:'
@ 04,19 say cname
store cname to mcname
select 3
use b:customer index b:custindx
find &mcname
    @ 05,02 say 'Customer Address:'
    @ 05,19 say address
select 2
@ 06,02 say 'Commodity:'
@ 06,19 say commodity
@ 07,02 say 'Price Base:'
@ 07,19 say pricebase
@ 08,02 say 'Time of Shipment      :      Months      Quantity '
@ 08,46 say '      Balance      Unitprice'

store 9 to mlnctr
select 4
use b:tmofship index b:tmshipdx
do while .not. eof()
    if snumber = mcontrno
        @ mlnctr,32 say month
        @ mlnctr,39 say wgt
        @ mlnctr,47 say bal
        @ mlnctr,57 say unitprice
        store mlnctr+1 to mlnctr
    endif
    skip
enddo
select 2
@ mlnctr+2,02 say 'Total shipment:'
@ mlnctr+2,22 say totalship
@ mlnctr+3,02 say 'Balance of ship:'
```

```
@ mlnctr+3,22 say balofship
@ mlnctr+4,02 say 'L/C number:'
@ mlnctr+4,22 say lcnumber
@ mlnctr+5,02 say 'L/C expire date:'
@ mlnctr+5,22 say lcexpdate
@ mlnctr+6,02 say 'L/C shipment date:'
@ mlnctr+6,22 say lcshipdate
@ mlnctr+7,02 say 'L/C amount:'
@ mlnctr+7,22 say lcamount
@ mlnctr+8,02 say 'L/C balance:'
@ mlnctr+8,22 say lcbal
@ mlnctr+9,02 say 'Issue Bank:'
@ mlnctr+9,22 say issuebank
@ mlnctr+10,02 say 'Draw Bank:'
@ mlnctr+10,22 say drawbank
@ mlnctr+11,02 say 'Miti number:'
@ mlnctr+11,22 say mitino
@ mlnctr+12,02 say '# of shipment made:'
@ mlnctr+12,22 say nofshipment
eject
return
* < End of SALEPRT.PRG >
```

* Program SALELIST.PRG ----- called from S_INQUERY.PRG

```
clear
store space(25) to mcname
select 2
use b:s_contract index b:s_conindx
find &mcontrno
@ 01,10 say '===== Feed Stuff Sale Contract Information ====='
@ 02,02 say 'Sale Number:'
@ 02,19 say snumber
@ 03,02 say 'Contract Date:'
@ 03,19 say contrdate
@ 04,02 say 'Customer Name:'
@ 04,19 say cname
store cname to mcname
select 3
use b:customer index b:custindx
find &mcname
    @ 05,02 say 'Customer Address:'
    @ 05,19 say address
select 2
@ 06,02 say 'Commodity:'
@ 06,19 say commodity
@ 07,02 say 'Price Base:'
@ 07,19 say pricebase
@ 08,02 say 'Time of Shipment      :      Months      Quantity '
@ 08,46 say '      Balance      Unitprice'

store 9 to mlnctr
select 4
use b:tmofship index b:tmshipdx
do while .not. eof()
    if snumber = mcontrno
        @ mlnctr,32 say month
        @ mlnctr,39 say wgt
        @ mlnctr,47 say bal
        @ mlnctr,57 say unitprice
        store mlnctr+1 to mlnctr
    endif
    skip
enddo
select 2
if mlnctr >= 12
    wait 'More informations on next page...' to memvar
```

```
store 0 to mlnctr
clear
endif
@ mlnctr+2,02 say 'Total shipment:'
@ mlnctr+2,22 say totalship
@ mlnctr+3,02 say 'Balance of ship:'
@ mlnctr+3,22 say balofship
@ mlnctr+4,02 say 'L/C number:'
@ mlnctr+4,22 say lcnumber
@ mlnctr+5,02 say 'L/C expire date:'
@ mlnctr+5,22 say lcexpdate
@ mlnctr+6,02 say 'L/C shipment date:'
@ mlnctr+6,22 say lcshipdate
@ mlnctr+7,02 say 'L/C amount:'
@ mlnctr+7,22 say lcamount
@ mlnctr+8,02 say 'L/C balance:'
@ mlnctr+8,22 say lcbal
@ mlnctr+9,02 say 'Issue Bank:'
@ mlnctr+9,22 say issuebank
@ mlnctr+10,02 say 'Draw Bank:'
@ mlnctr+10,22 say drawbank
@ mlnctr+11,02 say 'Miti number:'
@ mlnctr+11,22 say mitino
@ mlnctr+12,02 say '* of shipment made:'
@ mlnctr+12,22 say nofshipment
return
* < End of SALEPRT.PRG >
```

```

* Program CONTRACT.FMT ----- called from NEWSALE.PRG
@ 01,05 say date()
@ 01,15 say 'NEW SALE CONTRACT DATA ENTRY SCREEN'
@ 02,15 say '=====
@ 04,05 say 'Sale Number:      ' get msnumber pict;
                                'xxxxxxxxxxxxx'
@ 05,05 say 'Customer Name:    ' get mcname pict;
                                'xxxxxxxxxxxxxxxxxxxxxxxxxxxxx'
@ 06,05 say 'Customer Address: ' get mcustaddr pict;
                                'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx'
@ 07,05 say 'Customer Phone#:  ' get mphoneno pict;
                                '(xxx)xxx-xxxx'
@ 08,05 say 'Contract Date:    ' get mcontrdate pict;
                                'xx/xx/xx'
@ 09,05 say 'Commodity:        ' get mcommodity pict;
                                'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx'
@ 10,05 say 'Price Base:       ' get mpricebase pict;
                                'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
                                xx'
@ 12,05 say 'L/C Number:      ' get mlcnumber pict;
                                'xxxxxxxxxxxxx'
@ 13,05 say 'L/C Exp. Date:    ' get mlcexpdate pict;
                                'xx/xx/xx'
@ 14,05 say 'L/C Ship Date:    ' get mlcshipdate pict;
                                'xx/xx/xx'
@ 15,05 say 'L/C Bale:        ' get mlcbal pict;
                                '999999999999.99'
@ 16,05 say 'L/C Amount:      ' get mlcamount pict;
                                '999999999999.99'
@ 17,05 say 'Issue Bank:      ' get missuebank pict;
                                'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx'
@ 18,05 say 'Draw Bank:       ' get mdrawbank pict;
                                'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx'
@ 19,05 say 'Miti Number:     ' get mmitino pict;
                                'xxxxxxxxxxxxxxxxxxxxx'
@ 23,05 say '--Leave all fields blank, and RETURN to exit--'
* < End of CONTRACT.FMT >

```

```

* Program SSHIPMENT.FMT ----- called from S_SHIPIN.PRG
@ 02,01 say date()
@ 02,15 say 'NEW SALE SHIPMENT DATA ENTRY SCREEN'
@ 03,15 say '===== '
@ 06,05 say 'Sale Number:      ' get msnumber pict;
                'xxxxxxxxxxxxx'
@ 07,05 say 'Customer Name: ' get mname pict;
                'xxxxxxxxxxxxxxxxxxxxxxxxxxxxx'
@ 08,05 say 'Origin:          ' get morigin pict;
                'xxxxxxxxxxxxxxxxxxxxxxxxxxxxx'
@ 09,05 say 'Destination:      ' get mdest pict;
                'xxxxxxxxxxxxxxxxxxxxxxxxxxxxx'
@ 10,05 say 'Estimate Date:      ' get metd pict 'xx/xx/xx'
@ 11,05 say 'Invoice Date:      ' get minvdate pict 'xx/xx/xx'
@ 12,05 say '===== Container Data Entry ====='
@ 13,05 say 'Invoice Number:    ' get minvoiceno pict;
                'xxxxxxxxxxxxxxxxxxxxx'
@ 14,05 say 'Container Number:  ' get mcnumber pict;
                'xxxxxxxxxxxxx'
@ 15,05 say 'Bales:            ' get mbales pict;
                '999999999999.99'
@ 16,05 say 'Net:              ' get mnet pict;
                '999999999999.99'
@ 23,05 say '--Leave all fields blank, and RETURN to exit--'
* < End of SSHIPMENT.FMT >

```

```

* Program TOFSHIP.FMT ----- called from NEWSALE.PRG
@ 01,05 say date()
@ 01,15 say 'TIME OF SHIPMENT DATA ENTRY SCREEN'
@ 02,15 say '=====
@ 04,05 say 'Month:      ' get mmonth pict '99'
@ 05,05 say 'Weight:    ' get mwgt pict '999999999999.99'
@ 06,05 say 'Bales:     ' get mbal pict '999999999999.99'
@ 07,05 say 'Unit Price: ' get munitprice pict '999999999999.99'
@ 23,05 say '--Leave all fields blank, and RETURN to exit--'
* < End of TOFSHIP.FMT >

```

```

* Program CONFMT.FMT ----- called from S_SHIPIN.PRG
@ 02,01 say date()
@ 02,15 say '          MORE CONTAINER DATA ENTRY SCREEN'
@ 03,15 say '          ====='
@ 06,05 say 'Container Number: ' get mcnumber pict;
          'xxxxxxxxxxxxx'
@ 07,05 say 'Invoice Number:   ' get minvoiceno pict;
          'xxxxxxxxxxxxxxxxx'
@ 08,05 say 'Bales:           ' get mbales pict;
          '999999999999.99'
@ 09,05 say 'Net:             ' get mnet pict;
          '999999999999.99'
@ 23,05 say '--Leave all fields blank, and RETURN to exit--'
* < End of CONFMT.FMT >

```

```

* Program PURCHASE ----- called from HANAOKA
set talk off
set bell off
set dele on
set exact on

do while .T.
clear
store ' ' to mchoice
@ 02,15 say '=====
@ 03,15 say '=
@ 04,15 say '=          HANAOKA Purchase Information      ='
@ 05,15 say '=
@ 06,15 say '=====
@ 08,15 say '          1. New Purchase Contract Entry'
@ 10,15 say '          2. New Purchase Shipment Entry'
@ 12,15 say '          3. Purchase Information Inquiry'
@ 14,15 say '          4. Quit'
@ 18,15 say '          Choose one function -->(1,2,3,4)' get mchoice;
                        pict 'x'

read
do case
    case mchoice = '1'
        do newpurch
    case mchoice = '2'
        do p_shipin
    case mchoice = '3'
        do p_inquer
    otherwise
        return
endcase
enddo
* < End of PURCHASE >

```

* Program NEWPURCH.PRG ----- called from PURCHASE.PRG

public mttotalship, mnofshipmnt, mpnumber

do while .T.

store	space(12)	to mpnumber
store	space(25)	to mfname
store	space(50)	to mcommodity, maddress
store	space(13)	to mphoneno
store	space(8)	to mcontrdate
store	0	to mttotalship
store	0	to mbalofship
store	0	to mnofshipmnt
store	.T.	to nogood, notok
store	' '	to manswer

do while notok

do while nogood

set confirm on

set format to contra_p

read

if mpnumber = ' '

set format to

set confirm off

use

close index

return

endif

use b:p_contract index b:p_conindx

find &mpnumber

if found()

? chr(7)

@ 06,50 say 'Duplicate key!!'

@ 24,10 say 'Replace? Discard? Inquiry?(R,D,I)-->';

get manswer pict 'x'

read

if upper(manswer) = 'R'

store .F. to nogood

@ 24,10 say ' '

endif

if upper(manswer) = 'I'

use

close index

do p_inquer

return

endif

```

    else
        store .F. to nogood
        set format to
    endif
enddo
store ' ' to manswer
@ 24,15 say 'Input OK?(Y/N)-->' get manswer pict 'x'
read
if upper(manswer) = 'Y'
    store .F. to notok
else
    store .T. to nogood
endif
enddo

do tmship
use
close index
use b:p_contract index b:p_conindx
    append blank
    replace pnumber           with mpnumber
    replace fname             with mfname
    replace contrdate         with mcontrdate
    replace commodity         with mcommodity
    replace totalship         with mtotalship
    replace balofship         with mbalofship
    replace nofshipment       with mnofshipmnt
use
close index

use b:farmer index b:farmerdx
find &mfname
if .NOT. found()
    append blank
    replace fname             with mfname
    replace address           with maddress
    replace phoneno           with mphoneno
endif
use
close index
enddo
* < End of NEWPURCH.PRG >

```

* Program P_SHIPIN.PRG ----- called from PURCHASE.PRG

clear

store .T. to nogood

public mshipno, mpnumber, mcost, mtotalebales, mtotallnet

public mtruckcost, mcompute, mnet, mcutpntwgt, mrateacut

public mratebcut, mcost, mbales

do while .T.

clear

store space(12) to mshipno, mpnumber

store space(12) to mwgtticket, mcontrno

store space(8) to mmthday

store 0 to mbales, mnoftruck

store 0 to mnet, mcost, mratebcut

store 0 to mrateacut

store 0 to mtotalebales, mshipintons

store 0 to mtotallnet, mcutpntwgt

store 0 to mpayment, mtruckcost

store ' ' to mcompute, manswer

@ 12,30 say "

accept ' Contract number:' to mcontrno

if mcontrno = ' '

return

endif

select 2

use b:p_contract index b:p_conindx

find &mcontrno

if found()

clear

set confirm on

do while nogood

set format to pshipin

read

if mpshipno = ' '

set format to

set confirm off

return

endif

@ 23,15 say "

accept ' Input OK?(Y/N)-->' to manswer

if upper(manswer) = 'Y'

store ' ' to manswer

@ 24,15 say ' '

store .F. to nogood

```

do pcompute
use b:truck index b:truckndx
append blank
replace pshipno           with mpshipno
replace pnumber           with mcontrno
replace mthday            with mmthday
store   space(8)          to   mmthday
replace wgtticketno       with mwgtticket
store   space(12)         to   mwgtticket
replace bales             with mbales
store   mtotalbales+mbales to   mtotalbales
store   0                 to   mbales
replace net               with mnet
store   mtotalnet+mnet    to   mtotalnet
store   0                 to   mnet
replace cost              with mcost
store   mtruckcost+mcost  to   mtruckcost
store   0                 to   mcost
use
close index
@ 23,15 say ''
accept '                  More trucks?(Y/N)-->' to manswer
if upper(manswer) = 'Y'
do truckin
endif
endif
enddo
use
close index

store mtotalnet/2000.0 to mshipintons
use b:tofship index b:tshipndx
do while .NOT. eof()
if pnumber=mcontrno
if month = substr(mthday,1,2)
store bal-mshipintons to bal
store mshipintons*unitprice to mpayment
endif
endif
skip
enddo
use
close index
use b:p_contract index b:p_conindx
do while .NOT. eof()

```

```

    if pnumber = mcontrno
        store balofship-mshipintons to balofship
    endif
    skip
enddo
use
close index
use b:p_shipment index b:pshipndx
append blank

replace pshipno           with mpshipno
replace pnumber           with mcontrno
replace noftruck          with mnoftruck
replace totalbales        with mtotalbales
replace totalnet          with mtotalnet
replace payment           with mpayment
replace truckcost         with mtruckcost
use
close index
else
    clear
    ? chr(7)
    @ 12,25 say 'No such contract in the file !!'
    @ 24,15
    wait
endif
enddo
*   < End of P_SHIPIN.PRG >

```

```

* Program TRUCKIN.PRG ----- called from P_SHIPIN.PRG
use
close index
use b:truck index b:truckndx
store .T. to mmore
do while mmore
clear
store ' ' to manswer
set confirm on
set format to truckfmt
read
if mwggticket = '
    set format to
    set confirm off
    use
    close index
    return
endif
@ 23,15 say "
accept '
                                Input OK?(Y/N)-->' to manswer

if upper(manswer) = 'Y'
    store ' ' to manswer
    append blank
    replace pshipno          with mpshipno
    replace pnumber          with mpnumber
    replace mthday           with mmthday
    replace wgtticketno      with mwggticket
    repalce bales            with mbales
    replace net              with mnet

do pcompute

    replace cost             with mcost
    store mtotalbales+mbales to mtotalbales
    store 0                  to mbales
    store mtotalnet+mnet     to mtotalnet
    store 0                  to mnet
    store mnoftruck+1       to mnoftruck
    store mtruckcost+mcost   to mtruckcost

@ 24,15 say '
@ 23,15 say "
accept '
                                More trucks?(Y/N)-->' to manswer

```

```
    if upper(manswer) = 'N'  
        store .F. to mmore  
        @ 24,15 say '  
    endif  
    store ' ' to manswer  
endif  
enddo  
use  
close index  
return  
* < End of TRUCKIN.PRG >
```

```

* Program P_INQUER.PRG ----- called from HANAOKA.PRG or
*                               from NEWPURCH.PRG or
*                               from P_SHIPIN.PRG

```

```

public mcontrno
do while .T.
clear store ' ' to mchoice, manswer
store ' ' to mcommoname
store ' ' to mcontrno, mshipno
@ 02,15 say '=====
@ 03,15 say '='
@ 04,15 say 'Purchase Information Inquiry ='
@ 05,15 say '='
@ 06,15 say '=====
@ 08,15 say '1. List all commodities'
@ 10,15 say '2. List all contract of one farmer'
@ 12,15 say '3. List one purchase contract info.'
@ 14,15 say '4. Quit'
@ 20,15 say 'Choose one function -->(1,2,3,4)' get;
mchoice pict 'x'

read
do case
case mchoice = '1'
clear
@ 02,15 say '==== List of all commodities ====
use b:p_contract index b:p_conindx
display all commodity off
wait
use
close index
case mchoice = '2'
clear
@ 02,15 say "Farmer's name:" get mcommoname pict;
'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx'
read
@ 03,15 say 'Contract with '
@ 03,32 say mcommoname
use b:p_contract index b:p_conindx
display all pnumber, commodity for fname=mcommoname;
off
wait
use
close index

```

```

case mchoice = '3'
  clear
  @ 02,15 say 'Contract Number:' get mcontrno pict;
                        'xxxxxxxxxxxxx'
  read
  if mcontrno = '      '
    return
  endif
  use b:p_contract index b:p_conindx
  find &mcontrno
  if found()
    use
    close index
    do purlist
    @ 24,15 say 'Need a printout?(Y/N)-->' get manswer;
                        pict 'x'
    read
    if upper(manswer) = 'Y'
      @ 24,15 say 'Turn on the printer,'
      wait
      set device to print
      do purlist
      set device to screen
    endif
    store ' ' to manswer
    use
    close index
    clear
    @ 12,15 say 'See all shipments in sequence?(Y/N)';
                        get manswer pict 'x'
    read
    if upper(manswer) = 'Y'
      do pshlist
      @ 24,15 say 'Need a printout?(Y/N)' get manswer;
                        pict 'x'
      read
      if upper(manswer) = 'Y'
        set device to print
        do pshlist
        set device to screen
      endif
    endif
    store ' ' to manswer
  else
    @ 15,20 say 'No such contract in the file!!'
  endif
endcase

```

```
        ? chr(7)
    endif
    use
    close index
    case mchoice = '4'
        return
    endcase
enddo
* < End of P_INQUER.PRG >
```

* Program TMSHIP.PRG ----- called from NEWPURCH.PRG

clear

store space(2) to mmonth

store 0 to mwgt,mbal,munitprice

store ' ' to manswer

store .T. to mcontinue

use b:tofship index b:tshipndx

do while mcontinue

set confirm on

set format to tofship

read

if mmonth = ' '

set format to

use

close index

return

else

? chr(7)

@ 21,15 say "

accept 'Input OK?(Y/N)-->' to manswer

if upper(manswer) = 'Y'

append blank

replace month

store ' '

replace pnumber

replace wgt

replace bal

store 0

replace unitprice

store 0

store ' '

store mtotalship+mwgt

store 0

store mnofshipmnt+1

with mmonth

to mmonth

with mpnumber

with mwgt

with mbal

to mbal

with munitprice

to munitprice

to manswer

to mtotalship

to mwgt

to mnofshipmnt

endif

endif

enddo

* < End of TMSHIP.PRG >

```

* program PURLIST.PRG ----- called from P_INQUER.PRG
clear
store space(25) to mfname
store ' ' to mlnctr
select 2
use b:p_contract index b:p_conindx
find &mcontrno
@ 01,02 say '=== Feed Stuff Purchase Contract Information ==='
@ 02,02 say 'Purchase number:'
@ 02,21 say pnumber
@ 03,02 say 'Contract date:'
@ 03,21 say contrdate
@ 04,02 say 'Farmer name:'
@ 04,21 say fname
store fname to mfname
select 3
use b:farmer index b:farmerdx
find &mfname
@ 05,02 say 'Address:'
@ 05,21 say address
select 2
@ 06,02 say 'Commodity:'
@ 06,21 say commodity
@ 07,02 say 'Time of shipment:  Month  Quantity  Balance
Unitprice'
store 8 to mlnctr
select 4
use b:tofship index b:tshipndx
do while .not. eof()
    if pnumber = mcontrno
        @ mlnctr,49 say unitprice
        @ mlnctr,35 say bal
        @ mlnctr,22 say wgt
        @ mlnctr,20 say month
        store mlnctr+1 to mlnctr
    endif
    skip
enddo
select 2
@ mlnctr+2,02 say 'Total Shipment:'
@ mlnctr+2,21 say totalship
@ mlnctr+3,02 say 'Balance of shipment:'
@ mlnctr+3,21 say balofship
@ mlnctr+4,02 say '* of shipments made:'

```

```

@ mlnctr+4,21 say nofshipment
eject
use
close index
return
* < End of PURLIST.PRG >

```

```

* Program PSHLIST.PRG ----- called from P_INQUER.PRG

```

```

clear
store 0 to mlnctr
store space(12) to mshipno
select 2
use b:p_shipment index b:pshipndx
select 3
use b:truck index b:truckndx
select 2
store .T. to notdone
do while .not. eof() .and. notdone
    if pnumber = mcontrno
        @ 01,02 say 'Shipment No:'
        @ 01,21 say pshipno
        store .F. to notdone
    endif
    skip
enddo
store pshipno to mpshipno
@ 02,02 say 'Date      Wgt tkt#      Bales      Netwgt      Cost'
store 3 to mlnctr
select 3
do while .not. eof()
    if pnumber = mcontrno
        @ mlnctr,01 say mthday
        @ mlnctr,09 say wgtticketno
        @ mlnctr,21 say bales
        @ mlnctr,40 say net
        @ mlnctr,55 say cost
        store mlnctr+1 to mlnctr
    endif
    skip
enddo

```

```
select 2
@ mlnctr+1,02 say 'Total Bales:'
@ mlnctr+1,21 say totlabales
@ mlnctr+2,02 say 'Total Net:'
@ mlnctr+2,21 say totalnet
@ mlnctr+3,02 say 'Payment:'
@ mlnctr+3,21 say payment
@ mlnctr+4,02 say 'Truck Cost:'
@ mlnctr+4,21 say truckcost
eject
use
close index
return
* < End of PSHLIST.PRG >
```

```

* Program PCOMPUTE.PRG ----- called from P_SHIPIN.PRG
if upper(mcompute) = 'S'
  if (mnet/2000.0) < mcutpntwgt
    store (mnet/2000.0)*mratebcut to mcost
  else
    store (mnet/2000.0)*mrateacut to mcost
  endif
else
  if mbales < mcutpntwgt
    store mratebcut to mcost
  else
    store (mbales/mcutpntwgt)*mrateacut to mcost
  endif
endif
return
* < End of PCOMPUTE.PRG >

```

```

* program CONTRA_P.FMT ----- called from NEWPURCH.PRG
@ 02,01 say date()
@ 02,15 say 'NEW PURCHASE CONTRACT DATA ENTRY SCREEN'
@ 03,15 say '=====
@ 06,05 say 'Purchase Number:' get mpnumber pict;
          'xxxxxxxxxxxxx'
@ 07,05 say 'Farmer name:      ' get mfname pict;
          'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx'
@ 08,05 say 'Farmer address:  ' get maddress pict;
          'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx'
@ 09,05 say 'Farmer phone#:    ' get mphoneno pict;
          '(xxx)xxx-xxxx'
@ 10,05 say 'Contract date:    ' get mcontrdate pict;
          'xx/xx/xx'
@ 11,05 say 'Commodity:        ' get mcommodity pict;
          'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx'
@ 12,05 say 'Total shipment:    ' get mttotalship pict;
          '999999999999.99'
@ 13,05 say 'Bales of shipment: ' get mbalofship pict;
          '999999999999.99'
@ 20,05 say '--Leave all fields blank, and <CR> to exit--'
* < End of CONTRA_P.FMT >

```

```

* Program PSHIPIN.FMT ----- called from P_SHIPIN.PRG
@ 02,01 say date()
@ 02,15 say 'NEW PURCHASE SHIPMENT DATA ENTRY SCREEN'
@ 03,15 say '===== '
@ 06,05 say 'Shipment Number:' get mshipno pict 'xxxxxxxxxxxxx'
@ 07,05 say 'Cutting point Wgt: ' get mcutpntwgt pict;
          '999999999999.99'
@ 08,05 say 'Rate below cut point:' get mratebcut pict;
          '999999999999.99'
@ 09,05 say 'Rate above cut point:' get mrateacut pict;
          '999999999999.99'
@ 13,15 say '      NEW TRUCKS DATA ENTRY SCREEN'
@ 14,15 say '===== '
@ 16,05 say 'Date:          ' get mmthday pict 'xx/xx/xx'
@ 17,05 say 'Wgt Ticket #: ' get mwggtticket pict 'xxxxxxxxxxxxx'
@ 18,05 say 'Bales:          ' get mbales pict '999999999999.99'
@ 19,05 say 'Net:            ' get mnet pict '999999999999.99'
@ 20,05 say 'Trucking rate computation by shorttons/bales(S/B)';
          get mcompute pict 'x'
@ 23,05 say '--Leave all fields blank, and <CR> to exit--'
* < End of PSHIPIN.FMT >

```

```

* Program TRUCKFMT.FMT ----- called from P_SHIPIN.PRG
@ 02,01 say date()
@ 02,15 say '          MORE TRUCKS DATA ENTRY SCREEN'
@ 03,15 say '=====
@ 05,05 say 'Date:          ' get mmthday pict 'xx/xx/xx'
@ 06,05 say 'Wgt Ticket #:  ' get mwggtticket pict 'xxxxxxxxxxxxx'
@ 07,05 say 'Bales:          ' get mbales pict '999999999999.99'
@ 08,05 say 'Net:            ' get mnet pict '999999999999.99'
@ 23,05 say '--Leave all fields blank, and <CR> to exit--'
*   < End of TRUCKFMT.FMT >

```

```

* Program TOFSHIP.FMT ----- called from NEWPURCH.PRG
@ 01,05 say date()
@ 01,15 say 'TIME OF SHIPMENT DATA ENTRY SCREEN'
@ 02,15 say '=====
@ 04,05 say 'Month:      ' get mmonth pict '99'
@ 05,05 say 'Weight:     ' get mwgt pict '999999999999.99'
@ 06,05 say 'Bales:      ' get mbal pict '999999999999.99'
@ 07,05 say 'Unit Price:  ' get munitprice '999999999999.99'
@ 23,05 say '--Leave all fields blank, and <CR> to exit--'
* < End of TOFSHIP.FMT >

```

LIST OF REFERENCES

1. Barston, David R., Interactive Programming Environment, McGraw Hill, Inc., 1984.
2. Weiderhold, Gio, Database Design, McGraw Hill, 1977.
3. Senn, James A., Information Systems in Management, Wadsworth Publishers, 1982.
4. Ullman, Jefferey D., Database Systems, Computer Software Engineering Series, 1982.
5. Luis Castro, Jay Hanson and Tom Retting., Advanced Programmer's Guide, featuring dBaseII and dBase III, Ashton Tate, 1985.
6. Bharucha, Kerman D., dBase III PLUS - A Comprehensive User's Manual, Tab Books Inc., 1986.
7. Jenkins, David., dBase III - Tips And Tricks, Hayden Book Company, 1986.
8. MacLennan, Bruce J., Principles Of Programming Languages : Design, Evaluation, and Implementation, Holt, Rinehart and Winston, 1983.
9. Koffman, Elliot B., Problem Solving And Structured Programming In Pascal, Addison Wesley Publishing Company, 1985.
10. Apple Computer, Apple II Instant Pascal Language Reference Manual, Addison Wesley Publishing Company Inc., 1985.
11. C. Wu, Thomas , Introduction To Database Systems, CS 4300 Course Notes, 1986.
12. C. Wu, Thomas , Advanced Database Systems, CS 4312 Course Notes, 1987.
13. Fairley, Richard, Software Engineering Concepts, MacGraw Hill Inc., 1985.

14. Date, C.J., An Introduction To Database Systems, Addison Wesley Publishing Company, 1986.
15. Kroenke, David, Database Processing : Fundamentals, Design, Implementation,
16. Inmon, William H., Effective Database Design, Prentice-Hall Inc., 1981.
17. Glass, Robert L., Noiseux Ronald A., Software Maintenance Guidebook, Prentice-Hall Inc., 1981.
18. Boehm et al., Characteristics of Software Quality, TRW Systems and Energy, Inc., North-Holland Publishing Company, 1978.
19. Computer Science & Technology, Performance Assurance and Data Integrity Practices, U.S. Department of Commerce, National Bureau of Standards, 1978.
20. Relue, Richard B., Comparison Of Microcomputer Based Database Management, Masters Thesis, Naval Postgraduate School, June, 1982.
21. Sivasankaran, T.R., IS 4183 Course Notes, 1987.

INITIAL DISTRIBUTION LIST

	No. Copies
1. Defense Technical Information Center Cameron Station Alexandria, Virginia 22304-6145	2
2. Chief Of Naval Operations Director, Information Systems (OP-945) Navy Department Washington, D.C. 20350-2000	2
3. Library, Code 0142 Naval Postgraduate School Monterey, California 93943- 5002	2
4. Computer Technology Curricular Office Naval Postgraduate School Code 37 Monterey, California 93943	1
5. Department Chairman Computer Science Department Code 52 Naval Postgraduate School Monterey, California 93943	1
6. Prof. C. Thomas Wu, Code 52 WQ Naval Postgraduate School Monterey, California 93943	5
7. Library, Chinese Naval Academy Tsoa-ying District Kaohsiung City Taiwan, Republic of China	2
8. Major To Chang No. 1-1, Lane 10, Kuo-Chan Rd. Feng Shan City Taiwan, Republic of China	3

9. Anne Liang
3981 Hamilton St. #4
San Diego, CA 92104

1

DUDLEY L. JONES
NAVAL POSTGRADUATE SCHOOL
MONTAGUE, CALIF. 94062-5000

Thesis

C3718 Chang

c.1 Comparison of Pascal
and the dBASE III PLUS
language in programming
an inventory management
system.

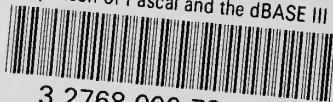
Thesis

C3718 Chang

c.1 Comparison of Pascal
and the dBASE III PLUS
language in programming
an inventory management
system.

thesC3718

Comparison of Pascal and the dBASE III P



3 2768 000 72984 2

DUDLEY KNOX LIBRARY